A

Abduction

Antonis C. Kakas
University of Cyprus, Nicosia, Cyprus

Definition

Abduction is a form of reasoning, sometimes described as "deduction in reverse," whereby given a rule that "*A follows from B*" and the observed result of "*A*" we infer the condition "*B*" of the rule. More generally, given a theory, *T*, modeling a domain of interest and an observation, "*A*," we infer a hypothesis "*B*" such that the observation follows deductively from *T* augmented with "*B*." We think of "*B*" as a possible explanation for the observation according to the given theory that contains our rule. This new information and its consequences (or ramifications) according to the given theory can be considered as the result of a (or part of a) learning process based on the given theory and driven by the observations that are explained by abduction. Abduction can be combined with ►induction in different ways to enhance this learning process.

Motivation and Background

Abduction is, along with induction, a *synthetic* form of reasoning whereby it generates, in its explanations, new information not hitherto contained in the current theory with which the reasoning is performed. As such, it has a natural relation to learning, and in particular to *knowledge intensive learning*, where the new information generated aims to complete, at least partially, the current knowledge (or model) of the problem domain as described in the given theory.

Early uses of abduction in the context of machine learning concentrated on how abduction can be used as a theory revision operator for identifying where the current theory could be revised in order to accommodate the new learning data. This includes the work of Michalski (1993), Ourston and Mooney (1994), and Ade, Malfait, and Raedt (1994). Another early link of abduction to learning was given by the ►explanation based learning method (DeJong & Mooney, 1986), where the abductive explanations of the learning data (training examples) are generalized to all cases.

Following this, it was realized (Flach & Kakas, 2000) that the role of abduction in learning could be strengthened by linking it to induction, culminating in a hybrid integrated approach to learning where abduction and induction are tightly integrated to provide powerful learning frameworks such as the ones of Progol 5.0 (Muggleton & Bryant, 2000) and HAIL (Ray, Broda, & Russo, 2003). On the other hand, from the point of view of abduction as "inference to the best explanation" (Josephson & Josephson, 1994) the link with induction provides a way to distinguish between different explanations and to select those explanations that give a better inductive generalization result.

A recent application of abduction, on its own or in combination with induction, is in Systems Biology where we try to model biological processes and pathways at different levels. This challenging domain provides an important development test-bed for these methods of knowledge intensive learning (see e.g., King et al., 2004; Papatheodorou, Kakas, & Sergot, 2005; Ray, Antoniades, Kakas, & Demetriades, 2006; Tamaddoni-Nezhad, Kakas, Muggleton, & Pazos, 2004; Zupan et al., 2003).

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Structure of the Learning Task

Abduction contributes to the learning task by first explaining, and thus rationalizing, the training data according to a given and current model of the domain to be learned. These abductive explanations either form on their own the result of learning or they feed into a subsequent phase to generate the final result of learning.

Abduction in Artificial Intelligence

Abduction as studied in the area of Artificial Intelligence and the perspective of learning is mainly defined in a logic-based approach (Other approaches to abduction include the set covering approach See, e.g., Reggia (1983) or case-based explanation, e.g., Leake (1995).) as follows.

Given a set of sentences T (a theory or model), and a sentence O (observation), the abductive task is the problem of finding a set of sentences H (abductive explanation for O) such that:

1. $T \cup H \models O$,
2. $T \cup H$ is consistent,

where $\models$ denotes the deductive entailment relation of the formal logic used in the representation of our theory and consistency refers also to the corresponding notion in this logic. The particular choice of this underlying formal framework of logic is in general a matter that depends on the problem or phenomena that we are trying to model. In many cases, this is based on ▶first order predicate calculus, as, for example, in the approach of theory completion in Muggleton and Bryant (2000). But other logics can be used, e.g., the nonmonotonic logics of default logic or logic programming with negation as failure when the modeling of our problem requires this level of expressivity.

This basic formalization as it stands, does not fully capture the explanatory nature of the abductive explanation H in the sense that it necessarily conveys some reason why the observations hold. It would, for example, allow an observation O to be explained by itself or in terms of some other observations rather than in terms of some "deeper" reason for which the observation must hold according to the theory T. Also, as the above specification stands, the observation can be abductively explained by generating in H some new (general) theory completely unrelated to the given theory T. In this case, H does not account for the observations O according to the given theory T and in this sense it may not be considered as an explanation for O relative to T. For these reasons, in order to specify a "level" at which the explanations are required and to understand these relative to the given general theory about the domain of interest, the members of an explanation are normally restricted to belong to a special preassigned, domain-specific class of sentences called *abducible*.

Hence abduction, is typically applied on a model, T, in which we can separate two disjoint sets of predicates: the *observable* predicates and the *abducible* (*or open*) predicates. The basic assumption then is that our model T has reached a sufficient level of comprehension of the domain such that all the incompleteness of the model can be isolated (under some working hypotheses) in its abducible predicates. The observable predicates are assumed to be completely defined (in T) in terms of the abducible predicates and other background auxiliary predicates; any incompleteness in their representation comes from the incompleteness in the abducible predicates. In practice, the empirical observations that drive the learning task are described using the observable predicates. Observations are represented by formulae that refer only to the observable predicates (and possibly some background auxiliary predicates) typically by ground atomic facts on these observable predicates. The abducible predicates describe underlying (theoretical) relations in our model that are not observable directly but can, through the model T, bring about observable information.

The assumptions on the abducible predicates used for building up the explanations may be subject to restrictions that are expressed through *integrity constraints*. These represent additional knowledge that we have on our domain expressing general properties of the domain that remain valid no matter how the theory is to be extended in the process of abduction and associated learning. Therefore, in general, an *abductive theory* is a triple, denoted by $\langle T, A, \mathrm{IC}\rangle$, where T is the background theory, A is a set of abducible predicates, and IC is a set of integrity constraints. Then, in the definition of an abductive explanation given above, one more requirement is added:

3. $T \cup H$ satisfies IC.

The satisfaction of integrity constraints can be formally understood in several ways (see Kakas, Kowalski, & Toni, 1992 and references therein). Note that the integrity constraints reduce the number of explanations for a set of observations filtering out those explanations that do not satisfy them. Based on this notion of abductive explanation a *credulous* form of abductive entailment is defined. Given an abductive theory, $T = \langle T, A, \text{IC} \rangle$, and an observation O then, O is *abductively entailed* by T, denoted by $T \models_A O$, if there exists an abductive explanation of O in T.

This notion of abductive entailment can then form the basis of a coverage relation for learning in the face of incomplete information.

Abductive Concept Learning

Abduction allows us to reason in the face of incomplete information. As such when we have learning problems where the background data on the training examples is incomplete the use of abduction can enhance the learning capabilities.

Abductive concept learning (ACL) (Kakas & Riguzzi, 2000) is a learning framework that allows us to learn from incomplete information and to later be able to classify new cases that again could be incompletely specified. Under ACL, we learn abductive theories, $\langle T, A, \text{IC} \rangle$ with abduction playing a central role in the covering relation of the learning problem. The abductive theories learned in ACL contain both rules, in T, for the concept(s) to be learned as well as general clauses acting as integrity constraints in IC.

Practical problems that can be addressed with ACL: (1) concept learning from incomplete background data where some of the background predicates are incompletely specified and (2) concept learning from incomplete background data together with given integrity constraints that provide some information on the incompleteness of the data. The treatment of incompleteness through abduction is integrated within the learning process. This allows the possibility of learning more compact theories that can alleviate the problem of over fitting due to the incompleteness in the data. A specific subcase of these two problems and important third application problem of ACL is that of (3) multiple predicate learning, where each predicate is required to be learned from the incomplete data for the other predicates. Here the abductive reasoning can be used to suitably connect and integrate the learning of the different predicates. This can help to overcome some of the nonlocality difficulties of multiple predicate learning, such as order-dependence and global consistency of the learned theory.

ACL is defined as an extension of ▶Inductive Logic Programming (ILP) where both the background knowledge and the learned theory are abductive theories. The central formal definition of ACL is given as follows where examples are atomic ground facts on the target predicate(s) to be learned.

Definition 1 (**Abductive Concept Learning**)
Given

- *A set of positive examples* E^+
- *A set of negative examples* E^-
- *An abductive theory* $T = \langle P, A, I \rangle$ *as background theory*
- *An hypothesis space* $\mathcal{T} = \langle \mathcal{P}, \mathcal{I} \rangle$ *consisting of a space of possible programs* $\mathcal{P}$ *and a space of possible constraints* $\mathcal{I}$

Find
A set of rules $P' \in \mathcal{P}$ *and a set of constraints* $I' \in \mathcal{I}$ *such that the new abductive theory* $T' = \langle P \cup P', A, I \cup I' \rangle$ *satisfies the following conditions*

- $T' \models_A E^+$
- $\forall e^- \in E^-,\ T' \not\models_A e^-$

where E^+ *stands for the conjunction of all positive examples.*

An individual example e is said to be *covered* by a theory T' if $T' \models_A e$. In effect, this definition replaces the deductive entailment as the example coverage relation in the ILP problem with abductive entailment to define the ACL learning problem.

The fact that the conjunction of positive examples must be covered means that, for every positive example, there must exist an abductive explanation and the explanations for all the positive examples must be consistent with each other. For negative examples, it is required that no abductive explanation exists for any of them. ACL can be illustrated as follows.

Example 2 *Suppose we want to learn the concept father. Let the background theory be* $T = \langle P, A, \varnothing \rangle$ *where*:

$P = \{parent(john, mary), male(john),$
$parent(david, steve),$
$parent(kathy, ellen), female(kathy)\},$
$A = \{male, female\}.$

Let the training examples be:

$E^+ = \{father(john, mary), father(david, steve)\},$
$E^- = \{father(kathy, ellen), father(john, steve)\}.$

In this case, a possible hypotheses $T' = \langle P \cup P', A, I' \rangle$ *learned by ACL would consist of*

$P' = \{father(X, Y) \leftarrow parent(X, Y), male(X)\},$
$I' = \{ \leftarrow male(X), female(X)\}.$

This hypothesis satisfies the definition of ACL because:

1. $T' \models_A father(john, mary), father(david, steve)$ *with* $\Delta = \{male(david)\}$.
2. $T' \not\models_A father(kathy, ellen)$, *as the only possible explanation for this goal, namely* $\{male(kathy)\}$ *is made inconsistent by the learned integrity constraint in* I'.
3. $T' \not\models_A father(john, steve)$, *as this has no possible abductive explanations.*

Hence, despite the fact that the background theory is incomplete (in its abducible predicates), ACL can find an appropriate solution to the learning problem by suitably extending the background theory with abducible assumptions. Note that the learned theory without the integrity constraint would not satisfy the definition of ACL, because there would exist an abductive explanation for the negative example $father(kathy, ellen)$, *namely* $\Delta^- = \{male(kathy)\}$. *This explanation is prohibited in the complete theory by the learned constraint together with the fact* $female(kathy)$.

The algorithm and learning system for ACL is based on a decomposition of this problem into two subproblems: (1) learning the rules in P' together with appropriate explanations for the training examples and (2) learning integrity constraints driven by the explanations generated in the first part. This decomposition allows ACL to be developed by combining the two IPL settings of explanatory (predictive) learning and confirmatory (descriptive) learning. In fact, the first subproblem can be seen as a problem of learning from entailment, while the second subproblem as a problem of learning from interpretations.

Abduction and Induction

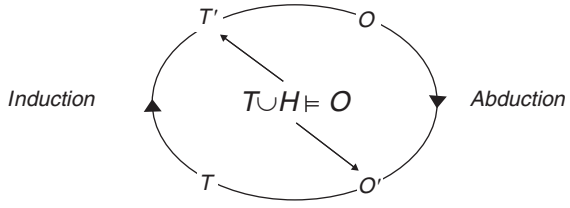
The utility of abduction in learning can be enhanced significantly when this is integrated with *induction*. Several approaches for synthesizing abduction and induction in learning have been developed, e.g., Ade and Denecker (1995), Muggleton and Bryant (2000), Yamamoto (1997), and Flach and Kakas (2000). These approaches aim to develop techniques for knowledge intensive learning with complex background theories. One problem to be faced by purely inductive techniques, is that the training data on which the inductive process operates, often contain gaps and inconsistencies. The general idea is that abductive reasoning can feed information into the inductive process by using the background theory for inserting new hypotheses and removing inconsistent data. Stated differently, abductive inference is used to complete the training data with hypotheses about missing or inconsistent data that explain the example or training data, using the background theory. This process gives alternative possibilities for assimilating and generalizing this data.

Induction is a form of synthetic reasoning that typically generates knowledge in the form of new general rules that can provide, either directly, or indirectly through the current theory T that they extend, new interrelationships between the predicates of our theory that can include, unlike abduction, the observable predicates and even in some cases new predicates. The inductive hypothesis thus introduces new, hitherto unknown, links between the relations that we are studying thus allowing new predictions on the observable predicates that would not have been possible before from the original theory under any abductive explanation.

An inductive hypothesis, H, extends, like in abduction, the existing theory T to a new theory $T' = T \cup H$, but now H provides new links between observables and nonobservables that was missing or incomplete in the original theory T. This is particularly evident from the fact that induction can be performed even with an empty given theory T, using just the set of observations. The observations specify incomplete (usually extensional) knowledge about the observable

predicates, which we try to *generalize* into new knowledge. In contrast, the generalizing effect of abduction, if at all present, is much more limited. With the given current theory T, that abduction always needs to refer to, we implicitly restrict the generalizing power of abduction as we require that the basic model of our domain remains that of T. Induction has a stronger and genuinely new generalizing effect on the observable predicates than abduction. While the purpose of abduction is to extend the theory with an explanation and then reason with it, thus enabling the generalizing potential of the given theory T, in induction the purpose is to extend the given theory to a new theory, which can provide new possible observable consequences.

This complementarity of abduction and induction – abduction providing explanations from the theory while induction generalizes to form new parts of the theory – suggests a basis for their integration within the context of theory formation and theory development. A *cycle of integration* of abduction and induction (Flach & Kakas, 2000) emerges that is suitable for the task of incremental modeling (Fig. 1). Abduction is used to transform (and in some sense normalize) the observations to information on the abducible predicates. Then, induction takes this as input and tries to generalize this information to general rules for the abducible predicates now treating these as observable predicates for its own purposes. The cycle can then be repeated by adding the learned information on the abducibles back in the model as new partial information



Abduction. Figure 1. The cycle of abductive and inductive knowledge development. The cycle is governed by the "equation" $T \cup H \models O$, where T is the current theory, O the observations triggering theory development, and H the new knowledge generated. On the left-hand side we have induction, its output feeding into the theory T for later use by abduction on the right; the abductive output in turn feeds into the observational data O' for later use by induction, and so on

on the incomplete abducible predicates. This will affect the abductive explanations of new observations to be used again in a subsequent phase of induction. Hence, through this cycle of integration the abductive explanations of the observations are added to the theory, not in the (simple) form in which they have been generated, but in a generalized form given by a process of induction on these.

A simple example, adapted from Ray et al. (2003), that illustrates this cycle of integration of abduction and induction is as follows. Suppose that our current model, T, contains the following rule and background facts:

sad(X) ← *tired(X), poor(X),*
tired(oli), tired(ale), tired(kr),
academic(oli), academic(ale), academic(kr),
student(oli), lecturer(ale), lecturer(kr),

where the only observable predicate is *sad*/1.

Given the observations $O = \{sad(ale), sad(kr),$ *not* $sad(oli)\}$ can we improve our model? The incompleteness of our model resides in the predicate *poor*. This is the only abducible predicate in our model. Using abduction we can explain the observations O via the explanation:

E = *{poor(ale), poor(kr), not poor(oli)}.*

Subsequently, treating this explanation as training data for inductive generalization we can generalize this to get the rule:

poor(X) ← *lecturer(X)*

thus (partially) defining the abducible predicate *poor* when we extend our theory with this rule.

This combination of abduction and induction has recently been studied and deployed in several ways within the context of ILP. In particular, *inverse entailment* (Muggleton and Bryant, 2000) can be seen as a particular case of integration of abductive inference for constructing a "bottom" clause and inductive inference to generalize it. This is realized in Progol 5.0 and applied to several problems including the discovery of the function of genes in a network of metabolic pathways (King et al., 2004), and more recently to the study of

inhibition in metabolic networks (Tamaddoni-Nezhad, Chaleil, Kakas, & Muggleton, 2006; Tamaddoni-Nezhad et al., 2004). In Moyle (2000), an ILP system called ALECTO, integrates a phase of *extraction-case abduction* to transform each case of a training example to an abductive hypothesis with a phase of induction that generalizes these abductive hypotheses. It has been used to learn robot navigation control programs by completing the specific domain knowledge required, within a general theory of planning that the robot uses for its navigation (Moyle, 2002).

The development of these initial frameworks that realize the cycle of integration of abduction and induction prompted the study of the problem of *completeness* for finding any hypotheses H that satisfies the basic task of finding a consistent hypothesis H such that $T \cup H \models O$ for a given theory T, and observations O. Progol was found to be incomplete (Yamamoto, 1997) and several new frameworks of integration of abduction and induction have been proposed such as SOLDR (Ito & Yamamoto, 1998), CF-induction (Inoue, 2001), and HAIL (Ray et al., 2003). In particular, HAIL has demonstrated that one of the main reasons for the incompleteness of Progol is that in its cycle of integration of abduction and induction, it uses a very restricted form of abduction. Lifting some of these restrictions, through the employment of methods from abductive logic programming (Kakas et al., 1992), has allowed HAIL to solve a wider class of problems. HAIL has been extended to a framework, called XHAIL (Ray, 2009), for learning nonmonotonic ILP, allowing it to be applied to learn Event Calculus theories for action description (Alrajeh, Ray, Russo, & Uchitel, 2009) and complex scientific theories for systems biology (Ray & Bryant, 2008).

Applications of this integration of abduction and induction and the cycle of knowledge development can be found in the recent proceedings of the Abduction and Induction in Artificial Intelligence workshops in 2007 (Flach & Kakas, 2009) and 2009 (Ray, Flach, & Kakas, 2009).

Abduction in Systems Biology

Abduction has found a rich field of application in the domain of systems biology and the declarative modeling of computational biology. In a project called, Robot scientist (King et al., 2004), Progol 5.0 was used to generate abductive hypotheses about the function of genes. Similarly, learning the function of genes using abduction has been studied in GenePath (Zupan et al., 2003) where experimental genetic data is explained in order to facilitate the analysis of genetic networks. Also in Papatheodorou et al. (2005) abduction is used to learn gene interactions and genetic pathways from microarray experimental data. Abduction and its integration with induction has been used in the study of inhibitory effect of toxins in metabolic networks (Tamaddoni-Nezhad et al., 2004, 2006) taking into account also the temporal variation that the inhibitory effect can have. Another bioinformatics application of abduction (Ray et al., 2006) concerns the modeling of human immunodeficiency virus (HIV) drug resistance and using this in order to assist medical practitioners in the selection of antiretroviral drugs for patients infected with HIV. Also, the recently developed frameworks of XHAIL and CF-induction have been applied to several problems in systems biology, see e.g., Ray (2009), Ray and Bryant (2008), and Doncescu, Inoue, and Yamamoto (2007), respectively.

Cross References

►Explanation-Based Learning
►Inductive Logic Programming

Recommended Reading

Ade, H., & Denecker, M. (1995). AILP: Abductive inductive logic programming. In C. S. Mellish (Ed.), *IJCAI* (pp. 1201–1209). San Francisco: Morgan Kaufmann.

Ade, H., Malfait, B., & Raedt, L. D. (1994). Ruth: An ILP theory revision system. In *ISMIS94*. Berlin: Springer.

Alrajeh, D., Ray, O., Russo, A., & Uchitel, S. (2009). Using abduction and induction for operational requirements elaboration. *Journal of Applied Logic*, *7*(3), 275–288.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternate view. *Machine Learning*, *1*, 145–176.

Doncescu, A., Inoue, K., & Yamamoto, Y. (2007). Knowledge based discovery in systems biology using cf-induction. In H. G. Okuno & M. Ali (Eds.), *IEA/AIE* (pp. 395–404). Heidelberg: Springer.

Flach, P., & Kakas, A. (2000). Abductive and inductive reasoning: Background and issues. In P. A. Flach & A. C. Kakas (Eds.), *Abductive and inductive reasoning. Pure and applied logic*. Dordrecht: Kluwer.

Flach, P. A., & Kakas, A. C. (Eds.). (2009). Abduction and induction in artificial intelligence [Special issue]. *Journal of Applied Logic, 7*(3).

Inoue, K. (2001). Inverse entailment for full clausal theories. In *LICS-2001 workshop on logic and learning*.

Ito, K., & Yamamoto, A. (1998). Finding hypotheses from examples by computing the least generlisation of bottom clauses. In *Proceedings of discovery science '98* (pp. 303–314). Berlin: Springer.

Josephson, J., & Josephson, S. (Eds.). (1994). *Abductive inference: Computation, philosophy, technology*. New York: Cambridge University Press.

Kakas, A., Kowalski, R., & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770.

Kakas, A., & Riguzzi, F. (2000). Abductive concept learning. *New Generation Computing*, *18*, 243–294.

King, R., Whelan, K., Jones, F., Reiser, P., Bryant, C., Muggleton, S., et al. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, *427*, 247–252.

Leake, D. (1995). Abduction, experience and goals: A model for everyday abductive explanation. *The Journal of Experimental and Theoretical Artificial Intelligence*, *7*, 407–428.

Michalski, R. S. (1993). Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning*, *11*, 111–151.

Moyle, S. (2002). Using theory completion to learn a robot navigation control program. In *Proceedings of the 12th international conference on inductive logic programming* (pp. 182–197). Berlin: Springer.

Moyle, S. A. (2000). *An investigation into theory completion techniques in inductive logic programming*. PhD thesis, Oxford University Computing Laboratory, University of Oxford.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Muggleton, S., & Bryant, C. (2000). Theory completion using inverse entailment. In *Proceedings of the tenth international workshop on inductive logic programming (ILP-00)* (pp. 130–146). Berlin: Springer.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, *66*, 311–344.

Papatheodorou, I., Kakas, A., & Sergot, M. (2005). Inference of gene relations from microarray data by abduction. In *Proceedings of the eighth international conference on logic programming and non-monotonic reasoning (LPNMR'05)* (Vol. 3662, pp. 389–393). Berlin: Springer.

Ray, O. (2009). Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, *7*(3), 329–340.

Ray, O., Antoniades, A., Kakas, A., & Demetriades, I. (2006). Abductive logic programming in the clinical management of HIV/AIDS. In G. Brewka, S. Coradeschi, A. Perini, & P. Traverso (Eds.), *Proceedings of the 17th European conference on artificial intelligence. Frontiers in artificial intelligence and applications* (Vol. 141, pp. 437–441). Amsterdam: IOS Press.

Ray, O., Broda, K., & Russo, A. (2003). Hybrid abductive inductive learning: A generalisation of Progol. In *Proceedings of the 13th international conference on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 2835, pp. 311–328). Berlin: Springer.

Ray, O., & Bryant, C. (2008). Inferring the function of genes from synthetic lethal mutations. In *Proceedings of the second international conference on complex, intelligent and software intensive systems* (pp. 667–671). Washington, DC: IEEE Computer Society.

Ray, O., Flach, P. A., & Kakas, A. C. (Eds.). (2009). Abduction and induction in artificial intelligence. *Proceedings of IJCAI 2009 workshop*.

Reggia, J. (1983). Diagnostic experts systems based on a set-covering model. *International Journal of Man-Machine Studies*, *19*(5), 437–460.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., & Muggleton, S. (2006). Application of abductive ILP to learning metabolic network inhibition from temporal data. *Machine Learning*, *64*(1–3), 209–230.

Tamaddoni-Nezhad, A., Kakas, A., Muggleton, S., & Pazos, F. (2004). Modelling inhibition in metabolic pathways through abduction and induction. In *Proceedings of the 14th international conference on inductive logic programming* (pp. 305–322). Berlin: Springer.

Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In *Proceedings of the seventh international workshop on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1297, pp. 296–308). Berlin: Springer.

Zupan, B., Bratko, I., Demsar, J., Juvan, P., Halter, J., Kuspa, A., et al. (2003). Genepath: A system for automated construction of genetic networks from mutant data. *Bioinformatics*, *19*(3), 383–389.

Absolute Error Loss

►Mean Absolute Error

Accuracy

Definition

Accuracy refers to a measure of the degree to which the predictions of a ►model match the reality being modeled. The term *accuracy* is often applied in the context of ►classification models. In this context, *accuracy* = $P(\lambda(X) = Y)$, where XY is a ►joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes, this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

The accuracy of a model is often assessed or estimated by applying it to test data for which the ►labels (Y values) are known. The accuracy of a classifier on test data may be calculated as *number of correctly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ►Laplace estimate or an ►*m*-estimate.

Accuracy is directly related to ►error rate, such that *accuracy* = 1.0 − *error rate* (or when expressed as a percentage, *accuracy* = 100 − *error rate*).

Cross References

►Confusion Matrix
►Resubstitution Accuracy

ACO

►Ant Colony Optimization

Actions

In a ►Markov decision process, *actions* are the available choices for the decision-maker at any given *decision epoch*, in any given *state*.

Active Learning

David Cohn
Mountain View, CA, USA

Definition

The term *Active Learning* is generally used to refer to a learning problem or system where the learner has some role in determining on what data it will be trained. This is in contrast to *Passive Learning*, where the learner is simply presented with a ►training set over which it has no control. Active learning is often used in settings where obtaining ►labeled data is expensive or time-consuming; by sequentially identifying which examples are most likely to be useful, an active learner can sometimes achieve good performance, using far less ►training data than would otherwise be required.

Structure of Learning System

In many machine learning problems, the training data are treated as a fixed and given part of the problem definition. In practice, however, the training data are often not fixed beforehand. Rather, the learner has an opportunity to play a role in deciding what data will be acquired for training. This process is usually referred to as "active learning," recognizing that the learner is an active participant in the training process.

The typical goal in active learning is to select training examples that best enable the learner to minimize its loss on future test cases. There are many theoretical and practical results demonstrating that, when applied properly, active learning can greatly reduce the number of training examples, and even the computational effort required for a learner to achieve good generalization.

A toy example that is often used to illustrate the utility of active learning is that of learning a threshold function over a one-dimensional interval. Given +/− labels for N points drawn uniformly over the interval, the expected error between the true value of the threshold and any learner's best guess is bounded by $O(1/N)$. Given the opportunity to sequentially select the position of points to be labeled, however, a learner can pursue a binary search strategy, obtaining a best guess that is within $O(1/2^N)$ of the true threshold value.

This toy example illustrates the sequential nature of example selection that is a component of most (but not all) active learning strategies: the learner makes use of initial information to discard parts of the solution space, and to focus future data acquisition on distinguishing parts that are still viable.

Related Problems

The term "active learning" is usually applied in supervised learning settings, though there are many related problems in other branches of machine learning and beyond. The "exploration" component of the "exploration/exploitation" strategy in reinforcement learning is one such example. The learner *must* take actions to gain information, and must decide what actions will give him/her the information that will best minimize future loss. A number of fields of Operations Research predate and parallel machine learning work on active learning, including Decision Theory (North, 1968), Value of Information Computation, Bandit problems (Robbins, 1952), and Optimal Experiment Design (Fedorov, 1972; Box & Draper, 1987).

Active Learning Scenarios

When active learning is used for classification or regression, there are three common settings: *constructive* active learning, *pool-based* active learning, and *stream-based* active learning (also called *selective sampling*).

Constructive Active Learning

In constructive active learning, the learner is allowed to propose arbitrary points in the input space as examples to be labeled. While this in theory gives the learner the most power to explore, it is often not practical. One obstacle is the observation that most learning systems train on only a reduced representation of the instances they are presented with: text classifiers on bags of words (rather than fully-structured text) and speech recognizers on formants (rather than raw audio). While a learning system may be able to identify what pattern of formants would be most informative to label, there is no reliable way to generate audio that a human could recognize (and label) from the desired formants alone.

Pool-Based Active Learning

Pool-based active learning (McCallum & Nigam, 1998) is popular in domains such as text classification and speech recognition where unlabeled data are plentiful and cheap, but labels are expensive and slow to acquire. In pool-based active learning, the learner may not propose arbitrary points to label, but instead has access to a set of unlabeled examples, and is allowed to select which of them to request labels for.

A special case of pool-based learning is transductive active learning, where the test distribution is exactly the set of unlabeled examples. The goal then is to sequentially select and label a small number of examples that will best allow predicting the labels of those points that remain unlabeled.

A theme that is common to both constructive and pool-based active learning is the principle of sequential experimentation. Information gained from early queries allows the learner to focus its search on portions of the domain that are most likely to give it additional information on subsequent queries.

Stream-Based Active Learning

Stream-based active learning resembles pool-based learning in many ways, except that the learner only has access to the unlabeled instances as a stream; when an instance arrives, the learner must decide whether to ask for its label or let it go.

Other Forms of Active Learning

By virtue of the broad definition of active learning, there is no real limit on the possible settings for framing it. Angluin's early work on learning regular sets (Angluin, 1987) employed a "counterexample" oracle: the learner would propose a solution, and the oracle would either declare it correct, or divulge a counterexample – an instance on which the proposed and true solutions disagreed. Jin and Si (2003) describe a Bayesian method for selecting informative items to recommend when learning a collaborative filtering model, and Steck and Jaakkola (2002) describe a method best described as *unsupervised* active learning to build Bayesian networks in large domains.

While most active learning work assumes that the cost of obtaining a label is independent of the instance to be labeled, there are many scenarios where this is not the case. A mobile robot taking surface measurements must first travel to the point it wishes to sample, making distant points more expensive than nearby ones. In some cases, the cost of a query (e.g., the difficulty of traveling to a remote point to sample it) may not even be known until it is made, requiring the learner to learn a model of that as well. In these situations, the sequential nature of active learning is greatly accentuated, and a learner faces the additional challenges of planning under uncertainty (see "Greedy vs. Batch Active Learning," below).

Common Active Learning Strategies

1. *Version space partitioning.* The earliest practical active learning work (Ruff & Dietterich, 1989; Mitchell, 1982) explicitly relied on ►version space partitioning. These approaches tried to select examples on which there was maximal disagreement between hypotheses in the current version space. When such examples were labeled, they would invalidate as large a portion of the version space as possible. A limitation of explicit version space approaches is that, in underconstrained domains, a learner may waste its effort differentiating portions of the version space that have little

effect on the classifier's predictions, and thus on its error.

2. *Query by Committee* (Seung, Opper, & Sompolinsky 1992). In query by committee, the experimenter trains an ensemble of models, either by selecting randomized starting points (e.g., in the case of a neural network) or by bootstrapping the training set. Candidate examples are scored based on disagreement among the ensemble models – examples with high disagreement indicate areas in the sample space that are underdetermined by the training data, and therefore potentially valuable to label. Models in the ensemble may be looked at as samples from the version space; picking examples where these models disagree is a way of splitting the version space.
3. *Uncertainty sampling* (Lewis & Gail, 1994). Uncertainty sampling is a heuristic form of statistical active learning. Rather than sampling different points in the version space by training multiple learners, the learner itself maintains an explicit model of uncertainty over its input space. It then selects for labeling those examples on which it is least confident. In classification and regression problems, uncertainty contributes directly to expected loss (as the variance component of the "error = bias + variance" decomposition), so that gathering examples where the learner has greatest uncertainty is often an effective loss-minimization heuristic. This approach has also been found effective for non-probabilistic models, by simply selecting examples that lie near the current decision boundary. For some learners, such as support vector machines, this heuristic can be shown to be an approximate partitioning of the learner's version space (Tong & Koller, 2001).
4. *Loss minimization* (Cohn, Ghahramani, & Jordan, 1996). Uncertainty sampling can stumble when parts of the learner's domain are inherently noisy. It may be that, regardless of the number of samples labeled in some neighborhood, it will remain impossible to accurately predict these. In these cases, it would be desirable to not only model the learner's uncertainty over arbitrary parts of its domain, but also to model what effect labeling any future example is expected to have on that uncertainty. For some learning algorithms it is feasible to explicitly compute such estimates (e.g., for locally-weighted regression and mixture models, these estimates may be computed in closed form). It is, therefore, practical to select examples that directly minimize the expected loss to the learner, as discussed below under "Statistical Active Learning."

Statistical Active Learning

Uncertainty sampling and direct loss minimization are two examples of *statistical* active learning. Both rely on the learner's ability to statistically model its own uncertainty. When learning with a statistical model, such as a linear regressor or a mixture of Gaussians (Dasgupta, 1999), the objective is usually to find model parameters that minimize some form of expected loss. When active learning is applied to such models, it is natural to also select training data so as to minimize that same objective. As statistical models usually give us estimates on the probability of (as yet) unknown values, it is often straightforward to turn this machinery upon itself to assist in the active learning process (Cohn et al., 1996). The process is usually as follows:

1. Begin by requesting labels for a small random subsample of the examples $x_1, x_2, \mathrm{K}, x_n x$ and fit our model to the labeled data.
2. For any x in our domain, a statistical model lets us estimate both the conditional expectation $\hat{y}(x)$ and $\sigma^2_{\hat{y}(x)}$, the variance of that expectation. We estimate our current loss by drawing a new random sample of unlabeled data, and computing the averaged $\sigma^2_{\hat{y}(x)}$.
3. We now consider a candidate point $\tilde{x}$, and ask what reduction in loss we would obtain if we had labeled it $\tilde{y}$. If we knew its label with certainty, we could simply add the point to the training set, retrain, and compute the new expected loss. While we do not know the true $\tilde{y}$, we could, in theory, compute the new expected loss for every possible $\tilde{y}$ and average those losses, weighting them by our model's estimate of $p(\tilde{y}|\tilde{x})$. In practice, this is normally unfeasible; however, for some statistical models, such as locally-weighted regression and mixtures of Gaussians, we can compute the distribution of $p(\tilde{y}|\tilde{x})$ and its effect on $\sigma^2_{\hat{y}(x)}$ in closed form (Cohn et al., 1996).

4. Given the ability to estimate the expected effect of obtaining label $\tilde{y}$ for candidate $\tilde{x}$, we repeat this computation for a sample of Mcandidates, and then request a label for the candidate with the largest expected decrease in loss. We add the newly-labeled example to our training set, retrain, and begin looking at candidate points to add on the next iteration.

The Need for Reference Distributions

Step (2) above illustrates a complication that is unique to active learning approaches. Traditional "passive" learning usually relies on the assumption that the distribution over which the learner will be tested is the same as the one from which the training data were drawn. When the learner is allowed to select its own training data, it still needs some form of access to the distribution of data on which it will be tested. A pool-based or stream-based learner can use the pool or stream as a proxy for that distribution, but if the learner is allowed (or required) to construct its own examples, it risks wasting all its effort on resolving portions of the solution space that are of no interest to the problem at hand.

A Detailed Example: Statistical Active Learning with LOESS

LOESS (Cleveland, Devlin, & Gross, 1988) is a simple form of locally-weighted regression using a kernel function. When asked to predict the unknown output y corresponding to a given input x, LOESS computes a ►linear regression over known (x, y) pairs, in which it gives pair (x_i, y_i) weight according to the proximity of x_i to x. We will write this weighting as a kernel function, $K(x_i, x)$, or simplify it to k_i when there is no chance of confusion.

In the active learning setting, we will assume that we have a large supply of unlabeled examples drawn from the test distribution, along with labels for a small number of them. We wish to label a small number more so as to minimize the mean squared error (MSE) of our model. MSE can be decomposed into two terms: squared ►bias and variance. If we make the (inaccurate but simplifying) assumption that LOESS is approximately unbiased for the problem at hand, minimizing MSE reduces to minimizing the variance of our estimates.

Given n labeled pairs, and a prediction to make for input x, LOESS computes the following covariance statistics around x:

$$\mu_x = \frac{\sum_i k_i x_i}{n}, \quad \sigma_x^2 = \frac{\sum_i k_i (x_i - \mu_x)^2}{n},$$
$$\sigma_{xy} = \frac{\sum_i k_i (x_i - \mu_x)(y_i - \mu_y)}{n}$$
$$\mu_y = \frac{\sum_i k_i y_i}{n}, \quad \sigma_y^2 = \frac{\sum_i k_i (y_i - \mu_y)^2}{n},$$
$$\sigma_{y|x}^2 = \sigma_y^2 - \frac{\sigma_{xy}}{\sigma_x^2}$$

We can combine these to express the conditional expectation of y (our estimate) and its variance as:

$$\hat{y} = \mu_y + \frac{\sigma_{xy}}{\sigma_x^2}(x - \mu_x),$$
$$\sigma_{\hat{y}}^2 = \frac{\sigma_{y|x}^2}{n^2}\left(\sum_i k_i^2 + \frac{(x-\mu_x)^2}{\sigma_x^2}\sum_i k_i^2 \frac{(x_i - \mu_x)^2}{\sigma_x^2}\right).$$

Our proxy for model error is the variance of our prediction, integrated over the test distribution $\left\langle \sigma_{\hat{y}}^2 \right\rangle$. As we have assumed a pool-based setting in which we have a large number of unlabeled examples from that distribution, we can simply compute the above variance over a sample from the pool, and use the resulting average as our estimate.

To perform statistical active learning, we want to compute how our estimated variance will change if we add an (as yet unknown) label $\tilde{y}$ for an arbitrary $\tilde{x}$. We will write this new expected variance as $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$. While we do not *know* what value $\tilde{y}$ will take, our model gives us an estimated mean $\hat{y}(\tilde{x})$ and variance $\sigma_{\hat{y}(\tilde{x})}^2$ for the value, as above. We can add this "distributed" y value to LOESS just as though it were a discrete one, and compute the resulting expectation $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$ in closed form. Defining $\tilde{k}$ as $K(\tilde{x}, x)$, we write:

$$\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle = \frac{\left\langle \tilde{\sigma}_{y|x}^2 \right\rangle}{(n+\tilde{k})^2}\left(\sum_i k_i^2 + \tilde{k}^2 + \frac{(x-\tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} \times \left(\sum_i k_i^2 \frac{(x_i - \tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} + \tilde{k}^2 \frac{(\tilde{x} - \tilde{\mu}_x)^2}{\tilde{\sigma}_x^2}\right)\right),$$

where the component expectations are computed as follows:

$$\left\langle \tilde{\sigma}^2_{y|x} \right\rangle = \left\langle \tilde{\sigma}^2_{y} \right\rangle - \frac{\left\langle \tilde{\sigma}^2_{xy} \right\rangle}{\tilde{\sigma}^2_{x}},$$

$$\left\langle \tilde{\sigma}^2_{y} \right\rangle = \frac{n\sigma^2_y}{n+\tilde{k}} + \frac{n\tilde{k}(\sigma^2_{y|\tilde{x}} + (\hat{y}(\tilde{x}) - \mu_y)^2)}{(n+\tilde{k})^2},$$

$$\tilde{\mu}_x = \frac{n\mu_x + \tilde{k}\tilde{x}}{n+\tilde{k}},$$

$$\left\langle \tilde{\sigma}_{xy} \right\rangle = \frac{n\sigma_{xy}}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x} - \mu_x)(\hat{y}(\tilde{x}) - \mu_y)}{(n+\tilde{k})^2},$$

$$\tilde{\sigma}^2_{x} = \frac{n\sigma^2_x}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x} - \mu_x)^2}{(n+\tilde{k})^2},$$

$$\left\langle \tilde{\sigma}^2_{xy} \right\rangle = \left\langle \tilde{\sigma}_{xy} \right\rangle^2 + \frac{n^2\tilde{k}^2\sigma^2_{y|\tilde{x}}(\tilde{x} - \mu_x)^2}{(n+\tilde{k})^4}.$$

Greedy Versus Batch Active Learning

It is also worth pointing out that virtually all active learning work relies on greedy strategies – the learner estimates what single example best achieves its objective, requests that one, retrains, and repeats. In theory, it is possible to plan some number of queries ahead, asking what point is best to label now, given that N-1 more labeling opportunities remain. While such strategies have been explored in Operations Research for very small problem domains, their computational requirements make this approach unfeasible for problems of the size typically encountered in machine learning.

There are cases where retraining the learner after every new label would be prohibitively expensive, or where access to labels is limited by the number of iterations as well as by the total number of labels (e.g., for a finite number of clinical trials). In this case, the learner may select a set of examples to be labeled on each iteration. This batch approach, however, is only useful if the learner is able to identify a set of examples whose expected contributions are non-redundant, which substantially complicates the process.

Cross References

►Active Learning Theory

Recommended Reading

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2), 87–106.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2, 319–342.

Box, G. E. P., & Draper, N. (1987). *Empirical model-building and response surfaces*. New York: Wiley.

Cleveland, W., Devlin, S., & Gross, E. (1988). Regression by local fitting. *Journal of Econometrics*, 37, 87–114.

Cohn, D., Atlas, L., & Ladner, R. (1990). Training connectionist networks with queries and selective sampling. In D. Touretzky (Ed.)., *Advances in neural information processing systems*. Morgan Kaufmann.

Cohn, D., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4, 129–145. http://citeseer.ist.psu.edu/321503.html

Dasgupta, S. (1999). Learning mixtures of Gaussians. *Foundations of Computer Science*, 634–644.

Fedorov, V. (1972). *Theory of optimal experiments*. New York: Academic Press.

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987). *On the learnability of Boolean formulae*, *Proceedings of the 19th annual ACM conference on theory of computing* (pp. 285–295). New York: ACM Press.

Lewis, D. D., & Gail, W. A. (1994). *A sequential algorithm for training text classifiers. Proceedings of the 17th annual international ACM SIGIR conference* (pp. 3–12). Dublin.

McCallum, A., & Nigam, K. (1998). Employing EM and pool-based active learning for text classification. In *Machine learning: Proceedings of the fifteenth international conference* (ICML'98) (pp. 359–367).

North, D. W. (1968). A tutorial introduction to decision theory. *IEEE Transactions Systems Science and Cybernetics*, 4(3).

Pitt, L., & Valiant, L. G. (1988). Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4), 965–984.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 55, 527–535.

Ruff, R., & Dietterich, T. (1989). What good are experiments? *Proceedings of the sixth international workshop on machine learning*. Ithaca, NY.

Seung, H. S., Opper, M., & Sompolinsky, H. (1992). Query by committee. In *Proceedings of the fifth workshop on computational learning theory* (pp. 287–294). San Mateo, CA: Morgan Kaufmann.

Steck, H., & Jaakkola, T. (2002). Unsupervised active learning in large domains. In Proceeding of the conference on uncertainty in AI. http://citeseer.ist.psu.edu/steck02unsupervised.html

Active Learning Theory

Sanjoy Dasgupta
University of California, San Diego, La Jolla, CA, USA

Definition

The term *active learning* applies to a wide range of situations in which a learner is able to exert some control over its source of data. For instance, when fitting a

regression function, the learner may itself supply a set of data points at which to measure response values, in the hope of reducing the variance of its estimate. Such problems have been studied for many decades under the rubric of *experimental design* (Chernoff, 1972; Fedorov, 1972). More recently, there has been substantial interest within the machine learning community in the specific task of actively learning binary classifiers. This task presents several fundamental statistical and algorithmic challenges, and an understanding of its mathematical underpinnings is only gradually emerging. This brief survey will describe some of the progress that has been made so far.

Learning from Labeled and Unlabeled Data

In the machine learning literature, the task of learning a classifier has traditionally been studied in the framework of *supervised learning*. This paradigm assumes that there is a training set consisting of data points x (from some set $\mathcal{X}$) and their labels y (from some set $\mathcal{Y}$), and the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that will accurately predict the labels of data points arising in the future. Over the past 50 years, tremendous progress has been made in resolving many of the basic questions surrounding this model, such as "how many training points are needed to learn an accurate classifier?"

Although this framework is now fairly well understood, it is a poor fit for many modern learning tasks because of its assumption that all training points automatically come labeled. In practice, it is frequently the case that the raw, abundant, easily obtained form of data is *unlabeled*, whereas labels must be explicitly procured and are expensive. In such situations, the reality is that the learner starts with a large pool of unlabeled points and must then strategically decide which ones it wants labeled: how best to spend its limited budget.

Example: Speech recognition. When building a speech recognizer, the unlabeled training data consists of raw speech samples, which are very easy to collect: just walk around with a microphone. For all practical purposes, an unlimited quantity of such samples can be obtained. On the other hand, the "label" for each speech sample is a segmentation into its constituent phonemes, and producing even one such label requires substantial human time and attention. Over the past decades, research labs and the government have expended an enormous amount of money, time, and effort on creating labeled datasets of English speech. This investment has paid off, but our ambitions are inevitably moving past what these datasets can provide: we would now like, for instance, to create recognizers for other languages, or for English in specific contexts. Is there some way to avoid more painstaking years of data labeling, to somehow leverage the easy availability of raw speech so as to significantly reduce the number of labels needed? This is the hope of active learning.

Some early results on active learning were in the *membership query* model, where the data is assumed to be *separable* (that is, some hypothesis h perfectly classifies all points) and the learner is allowed to query the label of *any* point in the input space $\mathcal{X}$ (rather than being constrained to a prespecified unlabeled set), with the goal of eventually returning the perfect hypothesis h^*. There is a significant body of beautiful theoretical work in this model (Angluin, 2001), but early experiments ran into some telling difficulties. One study (Baum & Lang, 1992) found that when training a neural network for handwritten digit recognition, the queries synthesized by the learner were such bizarre and unnatural images that they were impossible for a human to classify. In such contexts, the membership query model is of limited practical value; nonetheless, many of the insights obtained from this model carry over to other settings (Hanneke, 2007a).

We will fix as our standard model one in which the learner is *given* a source of unlabeled data, rather than being able to generate these points himself. Each point has an associated label, but the label is initially *hidden*, and there is a cost for revealing it. The hope is that an accurate classifier can be found by querying just a few labels, much fewer than would be required by regular supervised learning.

How can the learner decide which labels to probe? One option is to select the query points at random, but it is not hard to show that this yields the same label complexity as supervised learning. A better idea is to choose the query points *adaptively*: for instance, start by querying some random data points to get a rough sense of where the decision boundary lies, and then gradually refine the estimate of the boundary by specifically querying points in its immediate vicinity. In other

words, ask for the labels of data points whose particular positioning makes them especially informative. Such strategies certainly sound good, but can they be fleshed out into practical algorithms? And if so, do these algorithms work well in the sense of producing good classifiers with fewer labels than would be required by supervised learning?

On account of the enormous practical importance of active learning, there are a wide range of algorithms and techniques already available, most of which resemble the aggressive, adaptive sampling strategy just outlined, and many of which show promise in experimental studies. However, a big problem with this kind of sampling is that very quickly the set of labeled points no longer reflects the underlying data distribution. This makes it hard to show that the classifiers learned have good statistical properties (for instance, that they converge to an optimal classifier in the limit of infinitely many labels). This survey will only discuss methods that have proofs of statistical well-foundedness, and whose label complexity can be explicitly analyzed.

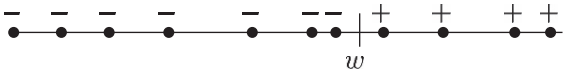
Motivating Examples

We will start by looking at a few examples that illustrate the enormous potential of active learning and that also make it clear why analyses of this new model require concepts and intuitions that are fundamentally different from those that have already been developed for supervised learning.

Example: Thresholds on the Line

Suppose the data lie on the real line, and the available classifiers are simple thresholding functions, $\mathcal{H} = \{h_w : w \in \mathbb{R}\}$:

$$h_w(x) = \begin{cases} +1 & \text{if } x \geq w \\ -1 & \text{if } x < w \end{cases}$$



To make things precise, let us denote the (unknown) underlying distribution on the data $(X, Y) \in \mathbb{R} \times \{+1, -1\}$ by $\mathbb{P}$, and let us suppose that we want a hypothesis $h \in \mathcal{H}$ whose error with respect to $\mathbb{P}$, namely $\text{err}_{\mathbb{P}}(h) = \mathbb{P}(h(X) \neq Y)$, is at most some ϵ. How many labels do we need?

In supervised learning, such issues are well understood. The standard machinery of sample complexity (using VC theory) tells us that if the data are *separable* – that is, if they can be perfectly classified by some hypothesis in $\mathcal{H}$ – then we need approximately $1/\epsilon$ random labeled examples from $\mathbb{P}$, and it is enough to return any classifier consistent with them.

Now suppose we instead draw $1/\epsilon$ *unlabeled* samples from $\mathbb{P}$. If we lay these points down on the line, their hidden labels are a sequence of −s followed by a sequence of +s, and the goal is to discover the point w at which the transition occurs. This can be accomplished with a simple binary search which asks for just $\log 1/\epsilon$ labels: first ask for the label of the median point; if it is +, move to the 25th percentile point, otherwise move to the 75th percentile point; and so on. Thus, for this hypothesis class, active learning gives an *exponential* improvement in the number of labels needed, from $1/\epsilon$ to just $\log 1/\epsilon$. For instance, if supervised learning requires a million labels, active learning requires just $\log 1{,}000{,}000 \approx 20$, literally!

It is a tantalizing possibility that even for more complicated hypothesis classes $\mathcal{H}$, a sort of generalized binary search is possible. A natural next step is to consider linear separators in *two* dimensions.

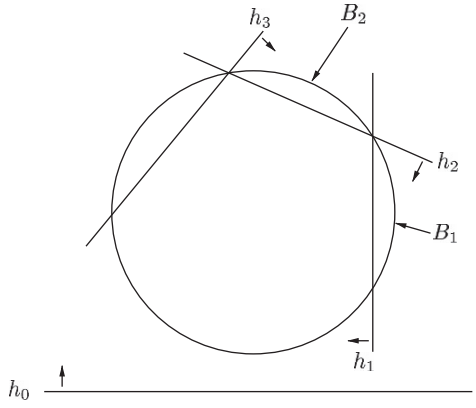
Example: Linear Separators in $\mathbb{R}^2$

Let $\mathcal{H}$ be the hypothesis class of linear separators in $\mathbb{R}^2$, and suppose the data is distributed according to some density supported on the perimeter of the unit circle. It turns out that the positive results of the one-dimensional case do not generalize: there are some target hypotheses in $\mathcal{H}$ for which $\Omega(1/\epsilon)$ labels are needed to find a classifier with error rate less than ϵ, no matter what active learning scheme is used.

To see this, consider the following possible target hypotheses (Fig. 1):

- h_0: all points are positive.
- h_i $(1 \leq i \leq 1/\epsilon)$: all points are positive except for a small slice B_i of probability mass ϵ.

The slices B_i are explicitly chosen to be disjoint, with the result that $\Omega(1/\epsilon)$ labels are needed to distinguish between these hypotheses. For instance, suppose nature chooses a target hypothesis at random from among the $h_i, 1 \leq i \leq 1/\epsilon$. Then, to identify this target with probability at least $1/2$, it is necessary to query points in at least (about) half the B_is.



Active Learning Theory. Figure 1. $\mathbb{P}$ **is supported on the circumference of a circle. Each** B_i **is an arc of probability mass** ϵ

Thus for these particular target hypotheses, active learning offers little improvement in sample complexity over regular supervised learning. What about other target hypotheses in $\mathcal{H}$, for instance those in which the positive and negative regions are more evenly balanced? It is quite easy (Dasgupta, 2005) to devise an active learning scheme which asks for $O(\min\{1/i(h), 1/\epsilon\}) + O(\log 1/\epsilon)$ labels, where $i(h) = \min\{\text{positive mass of } h, \text{negative mass of } h\}$. Thus even within this simple hypothesis class, the label complexity can run anywhere from $O(\log 1/\epsilon)$ to $\Omega(1/\epsilon)$, depending on the specific target hypothesis!

Example: An Overabundance of Unlabeled Data

In our two previous examples, the amount of unlabeled data needed was $O(1/\epsilon)$, exactly the usual sample complexity of supervised learning. But it is sometimes helpful to have significantly more unlabeled data than this. In Dasgupta (2005), a distribution $\mathbb{P}$ is described for which if the amount of unlabeled data is small (below any prespecified threshold), then the number of labels needed to learn the target linear separator is $\Omega(1/\epsilon)$; whereas if the amount of unlabeled data is much larger, then only $O(\log 1/\epsilon)$ labels are needed. This is a situation where most of the data distribution is fairly uninformative while a miniscule fraction is highly informative. A lot of unlabeled data is needed in order to get even a few of the informative points.

The Sample Complexity of Active Learning

We will think of the unlabeled points $x_1, \ldots, x_n$ as being drawn i.i.d. from an underlying distribution $\mathbb{P}_X$ on $\mathcal{X}$ (namely, the marginal of the distribution $\mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$), either all at once (a *pool*) or one at a time (a *stream*). The learner is only allowed to query the labels of points in the pool/stream; that is, it is restricted to "naturally occurring" data points rather than synthetic ones (Fig. 2). It returns a hypothesis $h \in \mathcal{H}$ whose quality is measured by its error rate, $\text{err}_{\mathbb{P}}(h)$.

In regular supervised learning, it is well known that if the VC dimension of $\mathcal{H}$ is d, then the number of labels that will with high probability ensure $\text{err}_{\mathbb{P}}(h) \leq \epsilon$ is roughly $O(d/\epsilon)$ if the data is separable and $O(d/\epsilon^2)$ otherwise (Haussler, 1992); various logarithmic terms are omitted here. For active learning, it is clear from the examples above that the VC dimension alone does not adequately characterize label complexity. Is there a different combinatorial parameter that does?

Generic Results for Separable Data

For separable data, it is possible to give upper and lower bounds on label complexity in terms of a special parameter known as the *splitting index* (Dasgupta, 2005). This is merely an existence result: the algorithm needed to realize the upper bound is intractable because it involves explicitly maintaining an ϵ-cover (a coarse approximation) of the hypothesis class, and the size of this cover is in general exponential in the VC dimension. Nevertheless, it does give us an idea of the kinds of label complexity we can hope to achieve.

Example. Suppose the hypothesis class consists of intervals on the real line: $\mathcal{X} = \mathbb{R}$ and $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{R}\}$, where $h_{a,b}(x) = \mathbf{1}(a \leq x \leq b)$. Using the splitting index, the label complexity of active learning is seen to be $\widetilde{\Theta}(\min\{1/\mathbb{P}_X([a,b]), 1/\epsilon\} + \log 1/\epsilon)$ when the target hypothesis is $h_{a,b}$ (Dasgupta, 2005). Here the $\widetilde{\Theta}$ notation is used to suppress logarithmic terms.

Example. Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{H}$ consists of linear separators through the origin. If $\mathbb{P}_X$ is the uniform distribution on the unit sphere, the number of labels needed to learn a hypothesis of error $\leq \epsilon$ is just $\widetilde{\Theta}(d \log 1/\epsilon)$, exponentially smaller than the $\widetilde{O}(d/\epsilon)$ label complexity of supervised learning. If $\mathbb{P}_X$ is not the uniform distribution but is everywhere within a multiplicative

Pool-based active learning

```
Get a set of unlabeled points U ⊂ 𝒳
Repeat until satisfied:
  Pick some x ∈ U to label
Return a hypothesis h ∈ ℋ
```

Stream-based active learning

```
Repeat for t = 0, 1, 2, ...:
  Choose a hypothesis h_t ∈ ℋ
  Receive an unlabeled point x ∈ 𝒳
  Decide whether to query its label
```

Active Learning Theory. Figure 2. Models of pool- and stream-based active learning. The data are draws from an underlying distribution $\mathbb{P}_X$, and hypotheses h are evaluated by $\text{err}_{\mathbb{P}}(h)$. If we want to get this error below ϵ, how many labels are needed, as a function of ϵ?

factor $\lambda > 1$ of it, then the label complexity becomes $\widetilde{O}((d \log 1/\epsilon) \log^2 \lambda)$, provided the amount of unlabeled data is increased by a factor of λ^2 (Dasgupta, 2005).

These results are very encouraging, but the question of an *efficient* active learning algorithm remains open. We now consider two approaches.

Mildly Selective Sampling

The label complexity results mentioned above are based on querying maximally informative points. A less aggressive strategy is to be *mildly selective*, to query all points except those that are quite clearly uninformative. This is the idea behind one of the earliest generic active learning schemes (Cohn, Atlas, & Ladner, 1994). Data points $x_1, x_2, \ldots$ arrive in a stream, and for each one the learner makes a spot decision about whether or not to request a label. When x_t arrives, the learner behaves as follows.

- Determine whether both possible labelings, $(x_t, +)$ and $(x_t, -)$, are consistent with the labeled examples seen so far.
- If so, ask for the label y_t. Otherwise set y_t to be the unique consistent label.

Fortunately, the check required for the first step can be performed efficiently by making two calls to a supervised learner. Thus this is a very simple and elegant active learning scheme, although as one might expect, it is suboptimal in its label complexity (Balcan et al., 2007). Interestingly, there is a parameter called the *disagreement coefficient* that characterizes the label complexity of this scheme and also of some other mildly selective learners (Friedman, 2009; Hanneke, 2007b).

In practice, the biggest limitation of the algorithm above is that it assumes the data are separable. Recent results have shown how to remove this assumption (Balcan, Beygelzimer, & Langford, 2006; Dasgupta et al., 2007) and to accommodate classification loss functions other than 0 – 1 loss (Beygelzimer et al., 2009). Variants of the disagreement coefficient continue to characterize label complexity in the agnostic setting (Beygelzimer et al., 2009; Dasgupta et al., 2007).

A Bayesian Model

The *query by committee* algorithm (Seung, Opper, & Sompolinsky, 1992) is based on a Bayesian view of active learning. The learner starts with a prior distribution on the hypothesis space, and is then exposed to a stream of unlabeled data. Upon receiving x_t, the learner performs the following steps.

- Draw two hypotheses h, h' at random from the posterior over $\mathcal{H}$.
- If $h(x_t) \neq h'(x_t)$ then ask for the label of x_t and update the posterior accordingly.

This algorithm queries points that substantially shrink the posterior, while at the same time taking account of the data distribution. Various theoretical guarantees have been shown for it (Freund, Seung, Shamir, & Tishby, 1997); in particular, in the case of linear separators with a uniform data distribution, it achieves a label complexity of $O(d \log 1/\epsilon)$, the best possible.

Sampling from the posterior over the hypothesis class is, in general, computationally prohibitive. However, for linear separators with a uniform prior, it can be implemented efficiently using random walks on convex bodies (Gilad-Bachrach, Navot, & Tishby, 2005).

Other Work

In this survey, I have touched mostly on active learning results of the greatest generality, those that apply to arbitrary hypothesis classes. There is also a significant body of more specialized results.

- Efficient active learning algorithms for specific hypothesis classes.
  This includes an online learning algorithm for linear separators that only queries some of the points and yet achieves similar regret bounds to algorithms that query all the points (Cesa-Bianchi, Gentile, & Zaniboni, 2004). The label complexity of this method is yet to be characterized.
- Algorithms and label bounds for linear separators under the uniform data distribution.
  This particular setting has been amenable to mathematical analysis. For separable data, it turns out that a variant of the perceptron algorithm achieves the optimal $O(d \log 1/\epsilon)$ label complexity (Dasgupta, Kalai, & Monteleoni, 2005). A simple algorithm is also available for the agnostic setting (Balcan et al., 2007).

Conclusion

The theoretical frontier of active learning is mostly an unexplored wilderness. Except for a few specific cases, we do not have a clear sense of how much active learning can reduce label complexity: whether by just a constant factor, or polynomially, or exponentially. The fundamental statistical and algorithmic challenges involved, together with the huge practical importance of the field, make active learning a particularly rewarding terrain for investigation.

Cross References

►Active Learning

Recommended Reading

Angluin, D. (2001). Queries revisited. In *Proceedings of the 12th international conference on algorithmic learning theory* (pp. 12–31).

Balcan, M.-F., Beygelzimer, A., & Langford, J. (2006). Agnostic active learning. In *International Conference on Machine Learning* (pp. 65–72). New York: ACM Press.

Balcan, M.-F., Broder, A., & Zhang, T. (2007). Margin based active learning. In *Conference on Learning Theory*. pp. 35–50.

Baum, E. B., & Lang, K. (1992). Query learning can work poorly when a human oracle is used. In *International Joint Conference on Neural Networks*.

Beygelzimer, A., Dasgupta, S., & Langford, J. (2009). Importance weighted active learning. In *International Conference on Machine Learning* (pp. 49–56). New York: ACM Press.

Cesa-Bianchi, N., Gentile, C., & Zaniboni, L. (2004). Worst-case analysis of selective sampling for linear-threshold algorithms. *Advances in Neural Information Processing Systems*.

Chernoff, H. (1972). Sequential analysis and optimal design. In *CBMS-NSF Regional Conference Series in Applied Mathematics 8*. SIAM.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning, 15*(2),201–221.

Dasgupta, S. (2005). Coarse sample complexity bounds for active learning. *Advances in Neural Information Processing Systems*.

Dasgupta, S., Kalai, A., & Monteleoni, C. (2005). Analysis of perceptron-based active learning. In *18th Annual Conference on Learning Theory*. pp. 249–263.

Dasgupta, S., Hsu, D. J., & Monteleoni, C. (2007). A general agnostic active learning algorithm. *Advances in Neural Information Processing Systems*.

Fedorov, V. V. (1972). *Theory of optimal experiments*. (W. J. Studden & E. M. Klimko, Trans.). New York: Academic Press.

Freund, Y., Seung, S., Shamir, E., & Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine Learning Journal, 28*,133–168.

Friedman, E. (2009). Active learning for smooth problems. In *Conference on Learning Theory*. pp. 343–352.

Gilad-Bachrach, R., Navot, A., & Tishby, N. (2005). Query by committeee made real. *Advances in Neural Information Processing Systems*.

Hanneke, S. (2007a). Teaching dimension and the complexity of active learning. In *Conference on Learning Theory*. pp. 66–81.

Hanneke, S. (2007b). A bound on the label complexity of agnostic active learning. In *International Conference on Machine Learning*. pp. 353–360.

Haussler, D. (1992). Decision-theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation, 100*(1),78–150.

Seung, H. S., Opper, M., & Sompolinsky, H. (1992). Query by committee. In *Conference on Computational Learning Theory*, pp. 287–294.

Adaboost

Adaboost is an ►ensemble learning technique, and the most well-known of the ►Boosting family of algorithms. The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models

should be able to compensate for errors made by earlier models. See ►ensemble learning for full details.

Adaptive Control Processes

►Bayesian Reinforcement Learning

Adaptive Real-Time Dynamic Programming

Andrew G. Barto
University of Massachusetts, Amherst, MA, USA

Synonyms

ARTDP

Definition

Adaptive Real-Time Dynamic Programming (ARTDP) is an algorithm that allows an agent to improve its behavior while interacting over time with an incompletely known dynamic environment. It can also be viewed as a heuristic search algorithm for finding shortest paths in incompletely known stochastic domains. ARTDP is based on ►Dynamic Programming (DP), but unlike conventional DP, which consists of off-line algorithms, ARTDP is an on-line algorithm because it uses agent behavior to guide its computation. ARTDP is adaptive because it does not need a complete and accurate model of the environment but learns a model from data collected during agent-environment interaction. When a good model is available, ►Real-Time Dynamic Programming (RTDP) is applicable, which is ARTDP without the model-learning component.

Motivation and Background

RTDP combines strengths of heuristic search and DP. Like heuristic search – and unlike conventional DP – it does not have to evaluate the entire state space in order to produce an optimal solution. Like DP – and unlike most heuristic search algorithms – it is applicable to nondeterministic problems. Additionally, RTDP's performance as an ►anytime algorithm is better than conventional DP and heuristic search algorithms. ARTDP extends these strengths to problems for which a good model is not initially available.

In artificial intelligence, control engineering, and operations research, many problems require finding a policy (or control rule) that determines how an agent (or controller) should generate actions in response to the states of its environment (the controlled system). When a "cost" or a "reward" is associated with each step of the agent's behavior, policies can be compared according to how much cost or reward they are expected to accumulate over time.

The usual formulation for problems like this in the discrete-time case is the ►Markov Decision Process (MDP). The objective is to find a policy that minimizes (maximizes) a measure of the total cost (reward) over time, assuming that the agent–environment interaction can begin in any of the possible states. In other cases, there is a designated set of "start states" that is much smaller than the entire state set (e.g., the initial board configuration in a board game). In these cases, any given policy only has to be defined for the set of states that can be reached from the starting states when the agent is using that policy. The rest of the states will never arise when that policy is being followed, so the policy does not need to specify what the agent should do in those states.

ARTDP and RTDP exploit situations where the set of states reachable from the start states is a small subset of the entire state space. They can dramatically reduce the amount of computation needed to determine an optimal policy for the relevant states as compared with the amount of computation that a conventional DP algorithm would require to determine an optimal policy for all the states. These algorithms do this by focussing computation around simulated behavioral experiences (if there is a model available capable of simulating these experiences), or around real behavioral experiences (if no model is available).

RTDP and ARTDP were introduced by Barto, Bradtke, and Singh (1995). The starting point was the novel observation by Bradtke that Korf's Learning Real-Time A* heuristic search algorithm (Korf, 1990)

is closely related to DP. RTDP generalizes Learning Real-Time A* to stochastic problems. ARTDP is also closely related to Sutton's Dyna system (Sutton, 1990) and Jalali and Ferguson's (1989) Transient DP. Theoretical analysis relies on the theory of Asnychronous DP as described by Bertsekas and Tsitsiklis (1989).

ARTDP and RTDP are ►model-based reinforcement learning algorithms, so called because they take advantage of an environment model, unlike ►model-free reinforcement learning algorithms such as ►Q-Learning and ►Sarsa.

Structure of Learning System

Backup Operations

A basic step of many DP and RL algorithms is a *backup operation*. This is an operation that updates a current estimate of the *cost* of an MDP's state. (We use the cost formulation instead of reward to be consistent with the original presentation of the algorithms. In the case of rewards, this would be called the *value* of a state and we would maximize instead of minimize.) Suppose X is the set of MDP states. For each state $x \in X$, $f(x)$, the cost of state x, gives a measure (which varies with different MDP formulations) of the total cost the agent is expected to incur over the future if it starts in x. If $f_k(x)$ and $f_{k+1}(x)$, respectively, denote the estimate of $f(x)$ before and after a backup, a typical backup operation applied to x looks like this:

$$f_{k+1}(x) = min_{a \in A}[c_x(a) + \sum_{y \in X} p_{xy}(a) f_k(y)],$$

where A is the set of possible agent actions, $c_x(a)$ is the immediate cost the agent incurs for performing action a in state x, and $p_{xy}(a)$ is the probability that the environment makes a transition from state x to state y as a result of the agent's action a. This backup operation is associated with the DP algorithm known as ►value iteration. It is also the backup operation used by RTDP and ARTDP.

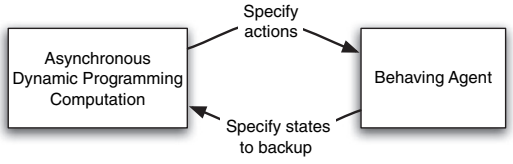
Conventional DP algorithms consist of successive "sweeps" of the state set. Each sweep consists of applying a backup operation to each state. Sweeps continue until the algorithm converges to a solution. Asynchronous DP, which underlies RTDP and ARTDP, does not use systematic sweeps. States can be chosen in any way whatsoever, and as long as backups continue to be applied to all states (and some other conditions are satisfied), the algorithm will converge. RTDP is an instance of asynchronous DP in which the states chosen for backups are determined by the agent's behavior.

The backup operation above is *model-based* because it uses known rewards and transition probabilities, and the values of all the states appear on the right-hand-side of the equation. In contrast, a *sample backup* uses the value of just one sample successor state. RTDP and ARTDP are like RL algorithms in that they rely on real or simulated behavioral experience, but unlike many (but not all) RL algorithms, they use full backups like DP.

Off-Line Versus On-Line

A conventional DP algorithm typically executes off-line. When applied to finding an optimal policy for an MDP, this means that the DP algorithm executes to completion before its result (an optimal policy) is used to control the agent's behavior. The sweeps of DP sequentially "visit" the states of the MDP, performing a backup operation on each state. But it is important not to confuse these visits with the behaving agent's visits to states: the agent is not yet behaving while the off-line DP computation is being done. Hence, the agent's behavior has no influence on the DP computation. The same is true for off-line asynchronous DP.

RTDP is an on-line, or "real-time," algorithm. It is an asynchronous DP computation that executes *concurrently* with the agent's behavior so that the agent's behavior can influence the DP computation. Further, the concurrently executing DP computation can influence the agent's behavior. The agent's visits to states directs the "visits" to states made by the concurrent asynchronous DP computation. At the same time, the action performed by the agent is the action specified by the policy corresponding to the latest results of the DP computation: it is the "greedy" action with respect to the current estimate of the cost function.



In the simplest version of RTDP, when a state is visited by the agent, the DP computation performs the

model-based backup operation given above on that same state. In general, for each step of the agent's behavior, RTDP can apply the backup operation to each of an arbitrary set of states, provided that the agent's current state is included. For example, at each step of behavior, a limited-horizon look-ahead search can be conducted from the agent's current state, with the backup operation applied to each of the states generated in the search. Essentially, RTDP is an asynchronous DP computation with the computational effort focused along simulated or actual behavioral trajectories.

Learning A Model

ARTDP is the same as RTDP except that (1) an environment model is updated using any on-line model-learning, or system identification, method, (2) the current environment model is used in performing the RTDP backup operations, and (3) the agent has to perform exploratory actions occasionally instead of always greedy actions as in RTDP. This last step is essential to ensure that the environment model eventually converges to the correct model. If the state and action sets are finite, the simplest way to learn a model is to keep counts of the number of times each transition occurs for each action and convert these frequencies to probabilities, thus forming the maximum-likelihood model.

Summary of Theoretical Results

When RTDP and ARTDP are applied to *stochastic optimal path problems*, one can prove that under certain conditions they converge to optimal policies without the need to apply backup operations to all the states. Indeed, is some problems, only a small fraction of the states need to be visited. A stochastic optimal path problem is an MDP with a nonempty set of start states and a nonempty set of goal states. Each transition until a goal state is reached has a nonnegative immediate cost, and once the agent reaches a goal state, it stays there and thereafter incurs zero cost. Each episode of agent experience begins with a start state. An optimal policy is one that minimizes the cost of every state, i.e., minimizes $f(x)$ for all states x. Under some relatively mild conditions, every optimal policy is guaranteed to eventually reach a goal state.

A state x is *relevant* if a start state s and an optimal policy exist such that x can be reached from s when the agent uses that policy. If we could somehow know which states are relevant, we could restrict DP to just these states and obtain an optimal policy. But this is not possible because knowing which states are relevant requires knowledge of optimal policies, which is what one is seeking. However, under certain conditions, without requiring repeated visits to all the irrelevant states, RTDP produces a policy that is optimal for all the relevant states. The conditions are that (1) the initial cost of every goal state is zero, (2) there exists at least one policy that guarantees that a goal state will be reached with probability one from any start state, (3) all immediate costs for transitions from non-goal states are strictly positive, and (4) none of the initial costs are larger than the actual costs. This result is proved in Barto et al. (1995) by combining aspects of Korf's (1990) proof for LRTA* with results for asynchronous DP.

Special Cases and Extensions

A number of special cases and extensions of RTDP have been developed that improve performance over the basic version. Some examples are as follows. Bonnet and Geffner's (2003) Labeled RTDP labels states that have already been "solved," allowing faster convergence than RTDP. Feng, Hansen, and Zilberstein (2003) proposed Symbolic RTDP, which selects a set of states to update at each step using symbolic model-checking techniques. The RTDP convergence theorem still applies because this is a special case of RTDP. Smith and Simmons (2006) developed Focused RTDP that maintains a priority value for each state to better direct search and produce faster convergence. Hansen and Zilberstein's (2001) LAO* uses some of the same ideas as RTDP to produce a heuristic search algorithm that can find solutions with loops to non-deterministic heuristic search problems. Many other variants are possible. Extending ARTDP instead of RTDP in all of these ways would produce analogous algorithms that could be used when a good model is not available.

Cross References

►Anytime Algorithm
►Approximate Dynamic Programming
►Reinforcement Learning
►System Identification

Recommended Reading

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1–2), 81–138.

Bertsekas, D., & Tsitsiklis, J. (1989). *Parallel and distributed computation: Numerical methods*. Englewood Cliffs, NJ: Prentice-Hall.

Bonet, B., & Geffner, H. (2003a). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the 13th international conference on automated planning and scheduling* (*ICAPS*-2003). Trento, Italy.

Bonet, B., & Geffner, H. (2003b). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the international joint conference on artificial intelligence* (*IJCAI*-2003). Acapulco, Mexico.

Feng, Z., Hansen, E., & Zilberstein, S. (2003). Symbolic generalization for on-line planning. In *Proceedings of the 19th conference on uncertainty in artificial intelligence.* Acapulco, Mexico.

Hansen. E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence, 129*, 35–62.

Jalali, A., & Ferguson, M. (1989). Computationally efficient control algorithms for Markov chains. In *Proceedings of the 28th conference on decision and control* (pp.1283–1288), Tampa, FL.

Korf, R. (1990). Real-time heuristic search. *Artificial Intelligence, 42*(2–3), 189–211.

Smith, T., & Simmons, R. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proceedings of the national conference on artificial intelligence (AAAI)*. Boston, MA: AAAI Press.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th international conference on machine learning* (pp.216–224). San Mateo, CA: Morgan Kaufmann.

Adaptive Resonance Theory

Gail A. Carpenter, Stephen Grossberg
Boston University, Boston, MA, USA

Synonyms

ART

Definition

Adaptive resonance theory, or ART, is both a cognitive and neural theory of how the brain quickly learns to categorize, recognize, and predict objects and events in a changing world, and a set of algorithms that computationally embody ART principles and that are used in large-scale engineering and technological applications wherein fast, stable, and incremental learning about complex changing environment is needed. ART clarifies the brain processes from which conscious experiences emerge. It predicts a functional link between processes of consciousness, learning, expectation, attention, resonance, and synchrony (CLEARS), including the prediction that "all conscious states are resonant states." This connection clarifies how brain dynamics enable a behaving individual to autonomously adapt in real time to a rapidly changing world. ART predicts how top-down attention works and regulates fast stable learning of recognition categories. In particular, ART articulates a critical role for "resonant" states in driving fast stable learning; and thus the name *adaptive resonance*. These resonant states are bound together, using top-down attentive feedback in the form of learned expectations, into coherent representations of the world. ART hereby clarifies one important sense in which the brain carries out predictive computation. ART has explained and successfully predicted a wide range of behavioral and neurobiological data, including data about human cognition and the dynamics of spiking laminar cortical networks. ART algorithms have been used in large-scale applications such as medical database prediction, remote sensing, airplane design, and the control of autonomous adaptive robots.

Motivation and Background

Many current learning algorithms do not emulate the way in which humans and other animals learn. The power of human and animal learning provides high motivation to discover computational principles whereby machines can learn with similar capabilities. Humans and animals experience the world on the fly, and carry out incremental learning of sequences of episodes in real time. Often such learning is unsupervised, with the world itself as the teacher. Learning can also proceed with an unpredictable mixture of unsupervised and supervised learning trials. Such learning goes on successfully in a world that is nonstationary; that is, the rules of which can change unpredictably through time. Moreover, humans and animals can learn quickly and stably through time. A single important experience can be remembered for a long time. ART proposes a solution of this *stability–plasticity dilemma* (Grossberg, 1980) by

showing how brains learn quickly without forcing catastrophic forgetting of already learned, and still successful, memories.

Thus, ART autonomously carries out fast, yet stable, incremental learning under both unsupervised and supervised learning conditions in response to a complex nonstationary world. In contrast, many current learning algorithms use batch learning in which all the information about the world to be learned is available at a single time. Other algorithms are not defined unless all learning trials are supervised. Yet other algorithms become unstable in a nonstationary world, or become unstable if learning is fast; that is, if an event can be fully learned on a single learning trial. ART overcomes these problems.

Some machine learning algorithms are feed-forward clustering algorithms that undergo catastrophic forgetting in a nonstationary world. The ART solution of the stability–plasticity dilemma depends upon feedback, or top-down, expectations that are matched against bottom-up data and thereby focus attention upon critical feature patterns. A good enough match leads to resonance and fast learning. A big enough mismatch leads to hypothesis testing or memory search that discovers and learns a more predictive category. Thus, ART is a self-organizing expert system that avoids the brittleness of traditional expert systems.

The world is filled with uncertainty, so probability concepts seem relevant to understanding how brains learn about uncertain data. This fact has led some machine learning practitioners to assume that brains obey Bayesian laws. However, the Bayes rule is so general that it can accommodate any system in nature. Additional computational principles and mechanisms must augment Bayes to distinguish a brain from, say, a hydrogen atom or storm. Moreover, probabilistic models often use nonlocal computations. ART shows how the brain embodies a novel kind of real-time probability theory, hypothesis testing, prediction, and decision-making, the local computations of which adapt to a nonstationary world. These ART principles and mechanisms go beyond Bayesian analysis, and are embodied parsimoniously in the laminar circuits of cerebral cortex. Indeed, the cortex embodies a new kind of laminar computing that reconciles the best properties of feedforward and feedback processing, digital and analog processing, and data-driven bottom-up processing combined with hypothesis-driven top-down processing (Grossberg, 2007).

Structure of Learning System

How CLEARS Mechanisms Interact

Humans are *intentional* beings who learn expectations about the world and make predictions about what is about to happen. Humans are also *attentional* beings who focus processing resources upon a restricted amount of incoming information at any time. Why are we both intentional and attentional beings, and are these two types of processes related? The stability–plasticity dilemma and its solution using resonant states provide a unifying framework for understanding these issues.

To clarify the role of sensory or cognitive expectations, and of how a resonant state is activated, suppose you were asked to "find the yellow ball as quickly as possible, and you will win a $10,000 prize." Activating an expectation of a "yellow ball" enables its more rapid detection, and with a more energetic neural response. Sensory and cognitive top-down expectations hereby lead to *excitatory matching* with consistent bottom-up data. Mismatch between top-down expectations and bottom-up data can suppress the mismatched part of the bottom-up data, to focus attention upon the matched, or expected, part of the bottom-up data.

Excitatory matching and attentional focusing on bottom-up data using top-down expectations generates resonant brain states: When there is a good enough match between bottom-up and top-down signal patterns between two or more levels of processing, their positive feedback signals amplify and prolong their mutual activation, leading to a resonant state. Amplification and prolongation of activity triggers learning in the more slowly varying adaptive weights that control the signal flow along pathways from cell to cell. Resonance hereby provides a global context-sensitive indicator that the system is processing data worthy of learning, hence the name *adaptive* resonance theory.

In summary, ART predicts a link between the mechanisms which enable us to learn quickly and stably about a changing world, and the mechanisms that enable us to learn expectations about such a world, test hypotheses about it, and focus attention upon information that we find interesting. ART clarifies this link by asserting that to solve the stability–plasticity

dilemma, only resonant states can drive rapid new learning.

It is just a step from here to propose that those experiences which can attract our attention and guide our future lives by being learned are also among the ones that are conscious. Support for this additional assertion derives from the many modeling studies whose simulations of behavioral and brain data using resonant states map onto properties of conscious experiences in those experiments.

The type of learning within the sensory and cognitive domain that ART mechanizes is *match learning*: Match learning occurs only if a good enough match occurs between bottom-up information and a learned top-down expectation that is read out by an active recognition category, or code. When such an approximate match occurs, previously learned knowledge can be refined. Match learning raises the concern about what happens if a match is not good enough? How does such a model escape perseveration on already learned representations?

If novel information cannot form a good enough match with the expectations that are read-out by previously learned recognition categories, then a memory search or hypothesis testing is triggered, which leads to selection and learning of a new recognition category, rather than catastrophic forgetting of an old one. Figure 1 illustrates how this happens in an ART model; it is discussed in great detail below. In contrast, learning within spatial and motor processes is proposed to be *mismatch learning* that continuously updates sensory-motor maps or the gains of sensory-motor commands. As a result, we can stably learn what is happening in a changing world, thereby solving the stability–plasticity dilemma, while adaptively updating our representations of where objects are and how to act upon them using bodies whose parameters change continuously through time. Brain systems that use inhibitory matching and mismatch learning cannot generate resonances; hence, their representations are not conscious.

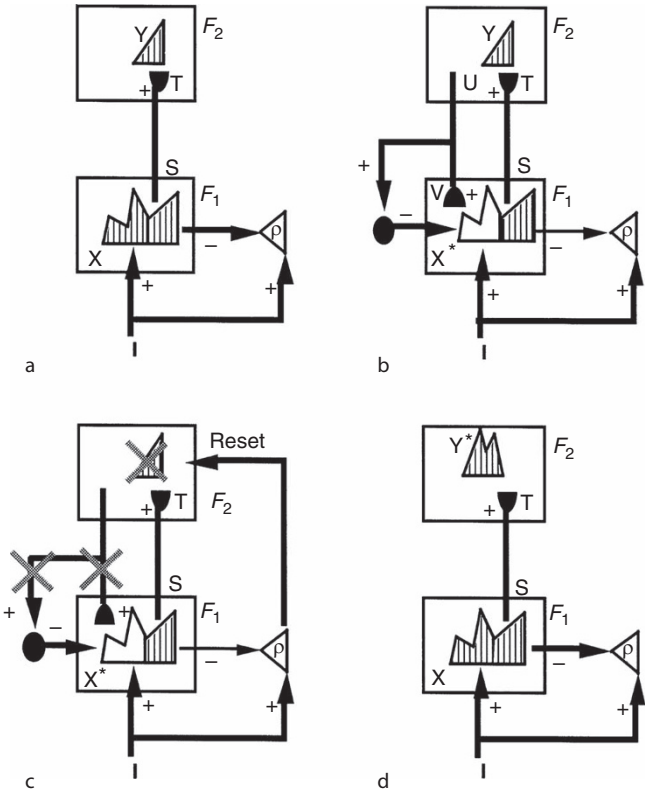
Complementary Computing in the Brain: Resonance and Reset

It has been mathematically proved that match learning within an ART model leads to stable memories in response to arbitrary list of events to be learned (e.g., Carpenter & Grossberg, 1987). However, match learning also has a serious potential weakness: If you can only learn when there is a good match between bottom-up data and learned top-down expectations, then how do you ever learn anything that you do not already know? ART proposes that this problem is solved by the brain by using an interaction between complementary processes of *resonance* and *reset,* which are predicted to control properties of attention and memory search, respectively. These complementary processes help our brains to balance between the complementary demands of processing the familiar and the unfamiliar, the expected and the unexpected.

Organization of the brain into complementary processes is predicted to be a general principle of brain design that is not just found in ART (Grossberg, 2000). A complementary process can individually compute some properties well, but cannot, by itself, process other complementary properties. In thinking intuitively about complementary properties, one can imagine puzzle pieces fitting together. Both pieces are needed to finish the puzzle. Complementary brain processes are more dynamic than any such analogy: Pairs of complementary processes interact to form emergent properties which overcome their complementary deficiencies to compute complete information with which to represent or control some aspect of intelligent behavior.

The resonance process in the complementary pair of resonance and reset is predicted to take place in the What cortical stream, notably in the inferotemporal and prefrontal cortex. Here top-down expectations are matched against bottom-up inputs. When a top-down expectation achieves a good enough match with bottom-up data, this match process focuses attention upon those feature clusters in the bottom-up input that are expected. If the expectation is close enough to the input pattern, then a state of resonance develops as the attentional focus takes hold.

Figure 1 illustrates these ART ideas in a simple two-level example. Here, a bottom-up input pattern, or vector, **I** activates a pattern X of activity across the feature detectors of the first level F_1. For example, a visual scene may be represented by the features comprising its boundary and surface representations. This feature pattern represents the relative importance of different features in the inputs pattern I. In Fig. 1a, the



Adaptive Resonance Theory. Figure 1. Search for a recognition code within an ART learning circuit: (a) The input pattern I is instated across the feature detectors at level F_1 as a short term memory (STM) activity pattern X. Input I also nonspecifically activates the orienting system with a gain that is called vigilance (ρ); that is, all the input pathways converge with gain ρ onto the orienting system and try to activate it. STM pattern X is represented by the hatched pattern across F_1. Pattern X both inhibits the orienting system and generates the output pattern *S*. Pattern *S* is multiplied by learned adaptive weights, also called long term memory (LTM) traces. These LTM-gated signals are added at F_2cells, or nodes, to form the input pattern *T*, which activates the STM pattern Y across the recognition categories coded at level F_2. (b) Pattern *Y* generates the top-down output pattern *U* which is multiplied by top-down LTM traces and added at F_1 nodes to form a *prototype* pattern *V* that encodes the learned expectation of the active F_2 nodes. Such a prototype represents the set of commonly shared features in all the input patterns capable of activating *Y*. If V mismatches I at F_1, then a new STM activity pattern X^* is selected *at* F_1. X^* is represented by the hatched pattern. It consists of the features of I that are confirmed by *V*. Mismatched features are inhibited. The inactivated nodes corresponding to unconfirmed features of X are unhatched. The reduction in total STM activity which occurs when X is transformed into X^* causes a decrease in the total inhibitionfrom F_1 to the orienting system. (c) If inhibition decreases sufficiently, the orienting system releases a nonspecific arousal wave to F_2; that is, a wave of activation that equally activates all F_2 nodes. This wave instantiates the intuition that "novel events are arousing." This arousal wave resets the STM pattern Y at F_2 by inhibiting *Y*. (d) After Y is inhibited, its top-down prototype signal is eliminated, and X can be reinstated at F_1. The prior reset event maintains inhibition of *Y* during the search cycle. As a result, X can activate a different STM pattern Y at F_2. If the top-down prototype due to this new *Y* pattern also mismatches I at F_1, then the search for an appropriate F_2 code continues until a more appropriate F_2 representation is selected. Such a search cycle represents a type of nonstationary hypothesis testing. When search ends, an attentive resonance develops and learning of the attended data is initiated (adapted with permission from Carpenter and Grossberg (1993)). The distributed ART architecture supports fast stable learning with arbitrarily distributed F_2 codes (Carpenter, 1997)

pattern peaks represent more activated feature detector cells, and the troughs, less-activated feature detectors. This feature pattern sends signals S through an adaptive filter to the second level F_2 at which a compressed representation Y (also called a recognition category, or a symbol) is activated in response to the distributed input T. Input T is computed by multiplying the signal vector S by a matrix of adaptive weights that can be altered through learning. The representation Y is compressed by competitive interactions across F_2 that allow only a small subset of its most strongly activated cells to remain active in response to T. The pattern Y in the figure indicates that a small number of category cells may be activated to different degrees. These category cells, in turn, send top-down signals U to F_1. The vector U is converted into the top-down expectation V by being multiplied by another matrix of adaptive weights. When V is received by F_1, a matching process takes place between the input vector **I** and **V** which selects that subset X* of F_1 features that were "expected" by the active F_2 category Y. The set of these selected features is the emerging "attentional focus."

Binding Distributed Feature Patterns and Symbols During Conscious Resonances

If the top-down expectation is close enough to the bottom-up input pattern, then the pattern X^* of attended features reactivates the category Y which, in turn, reactivates X^*. The network hereby locks into a resonant state through a positive feedback loop that dynamically links, or binds, the attended features across X^* with their category, or symbol, Y.

Resonance itself embodies another type of complementary processing. Indeed, there seem to be complementary processes both within and between cortical processing streams (Grossberg, 2000). This particular complementary relation occurs between distributed feature patterns and the compressed categories, or symbols, that selectively code them:

Individual features at F_1 have no meaning on their own, just like the pixels in a picture are meaningless one-by-one. The category, or symbol, in F_2 is sensitive to the global patterning of these features, and can selectively fire in response to this pattern. But it cannot represent the "contents" of the experience, including their conscious qualia, due to the very fact that a category is a compressed or "symbolic" representation. Practitioners of artificial intelligence have claimed that neural models can process distributed features, but not symbolic representations. This is not, of course, true in the brain. Nor is it true in ART.

Resonance between these two types of information converts the *pattern* of attended features into a coherent context-sensitive state that is linked to its category through feedback. This coherent state, which binds together distributed features and symbolic categories, can enter consciousness while it binds together spatially distributed features into either a stable equilibrium or a synchronous oscillation. The original ART article (Grossberg, 1976) predicted the existence of such synchronous oscillations, which were there described in terms of their mathematical properties as "order-preserving limit cycles." See Carpenter, Grossberg, Markuzon, Reynolds & Rosen (1992) and Grossberg & Versace (2008) for reviews of confirmed ART predictions, including predictions about synchronous oscillations.

Resonance Links Intentional and Attentional Information Processing to Learning

In ART, the resonant state, rather than bottom-up activation, is predicted to drive learning. This state persists long enough, and at a high enough activity level, to activate the slower learning processes in the adaptive weights that guide the flow of signals between bottom-up and top-down pathways between levels F_1 and F_2 in Fig. 1. This viewpoint helps to explain how adaptive weights that were changed through previous learning can regulate the brain's present information processing, without learning about the signals that they are currently processing unless they can initiate a resonant state. Through resonance as a mediating event, one can understand from a deeper mechanistic view why humans are intentional beings who are continually predicting what may next occur, and why we tend to learn about the events to which we pay attention.

More recent versions of ART, notably the synchronous matching ART (SMART) model (Grossberg & Versace, 2008) show how a match may lead to fast gamma oscillations that facilitate spike-timing dependent plasticity (STDP), whereas mismatch can lead to slower beta oscillations that lower the probability that mismatched events can be learned by a STDP learning law.

Complementary Attentional and Orienting Systems Control Resonance Versus Reset

A sufficiently bad mismatch between an active top-down expectation and a bottom-up input, say because the input represents an unfamiliar type of experience, can drive a memory search. Such a mismatch within the attentional system is proposed to activate a complementary *orienting system*, which is sensitive to unexpected and unfamiliar events. ART suggests that this orienting system includes the nonspecific thalamus and the hippocampal system. See Grossberg & Versace (2008) for a summary of data supporting this prediction. Output signals from the orienting system rapidly reset the recognition category that has been reading out the poorly matching top-down expectation (Figs. 1b and c). The cause of the mismatch is hereby removed, thereby freeing the system to activate a different recognition category (Fig. 1d). The reset event hereby triggers memory search, or hypothesis testing, which automatically leads to the selection of a recognition category that can better match the input.

If no such recognition category exists, say because the bottom-up input represents a truly novel experience, then the search process automatically activates an as yet uncommitted population of cells, with which to learn about the novel information. In order for a top-down expectation to match a newly discovered recognition category, its top-down adaptive weights initially have large values, which are pruned by the learning of a particular expectation.

This learning process works well under both unsupervised and supervised conditions (Carpenter et al., 1992). Unsupervised learning means that the system can learn how to categorize novel input patterns without any external feedback. Supervised learning uses predictive errors to let the system know whether it has categorized the information correctly. Supervision can force a search for new categories that may be culturally determined, and are not based on feature similarity alone. For example, separating the letters E and F that are of similar features into separate recognition categories is culturally determined. Such error-based feedback enables variants of E and F to learn their own category and top-down expectation, or prototype. The complementary, but interacting, processes of attentive-learning and orienting-search together realize a type of error correction through hypothesis testing that can build an ever-growing, self-refining internal model of a changing world.

Controlling the Content of Conscious Experiences: Exemplars and Prototypes

What combinations of features or other information are bound together into conscious object or event representations? One view is that exemplars or individual experiences are learned because humans can have very specific memories. For example, we can all recognize the particular faces of our friends. On the other hand, storing every remembered experience as exemplars can lead to a combinatorial explosion of memory, as well as to unmanageable problems of memory retrieval. A possible way out is suggested by the fact that humans can learn prototypes which represent general properties of the environment (Posner & Keele, 1968). For example, we can recognize that everyone has a face. But then how do we learn specific episodic memories? ART provides an answer that overcomes the problems faced by earlier models.

ART prototypes are not merely averages of the exemplars that are classified by a category, as is typically assumed in classical prototype models. Rather, they are the actively selected *critical feature patterns* upon which the top-down expectations of the category focus attention. In addition, the generality of the information that is codes by these critical feature patterns is controlled by a gain control process, called *vigilance* control, which can be influenced by environmental feedback or internal volition (Carpenter & Grossberg, 1987). Low vigilance permits the learning of general categories with abstract prototypes. High vigilance forces a memory search to occur for a new category when even small mismatches exist between an exemplar and the category that it activates. As a result, in the limit of high vigilance, the category prototype may encode an individual exemplar.

Vigilance is computed within the orienting system of an ART model (Fig. 1b–d). It is here that bottom-up excitation from all the active features in an input pattern I is compared with inhibition from all the active features in a distributed feature representation across F_1. If the ratio of the total activity across the active features in F_1 (i.e., the "matched" features) to the total activity of all the features in I is less than a *vigilance parameter* ρ (Fig. 1b), then a reset wave is activated (Fig. 1c), which

can drive the search for another category to classify the exemplar. In other words, the vigilance parameter controls how bad a match can be tolerated before search for a new category is initiated. If the vigilance parameter is low, then many exemplars can influence the learning of a shared prototype, by chipping away at the features that are not shared with all the exemplars. If the vigilance parameter is high, then even a small difference between a new exemplar and a known prototype (e.g., F vs. E) can drive the search for a new category with which to represent F.

One way to control vigilance is by a process of *match tracking*. Here a predictive error (e.g., D is predicted in response to F), the vigilance parameter increases until it is just higher than the ratio of active features in F_1 to total features in I. In other words, vigilance "tracks" the degree of match between input exemplar and matched prototype. This is the minimal level of vigilance that can trigger a reset wave and thus a memory search for a new category. Match tracking realizes a minimax learning rule that conjointly *maximizes* category generality while it *minimizes* predictive error. In other words, match tracking uses the least memory resources that can prevent errors from being made.

Because vigilance can vary across learning trials, recognition categories capable of encoding widely differing degrees of generalization or abstraction can be learned by a single ART system. Low vigilance leads to broad generalization and abstract prototypes. High vigilance leads to narrow generalization and to prototypes that represent fewer input exemplars, even a single exemplar. Thus a single ART system may be used, say, to learn abstract prototypes with which to recognize abstract categories of faces and dogs, as well as "exemplar prototypes" with which to recognize individual views of faces and dogs. ART models hereby try to learn the most general category that is consistent with the data. This tendency can, for example, lead to the type of overgeneralization that is seen in young children until further learning leads to category refinement.

Memory Consolidation and the Emergence of Rules: Direct Access to Globally Best Match

As sequences of inputs are practiced over learning trials, the search process eventually converges upon stable categories. It has been mathematically proved (Carpenter & Grossberg, 1987) that familiar inputs directly access the category whose prototype provides the best match globally, while unfamiliar inputs engage the orienting subsystem to trigger memory searches for better categories until they become familiar. This process continues until the memory capacity, which can be chosen arbitrarily large, is fully utilized. The process whereby search is automatically disengaged is a form of *memory consolidation* that emerges from network interactions. Emergent consolidation does not preclude structural consolidation at individual cells, since the amplified and prolonged activities that subserve a resonance may be a trigger for learning-dependent cellular processes, such as protein synthesis and transmitter production.

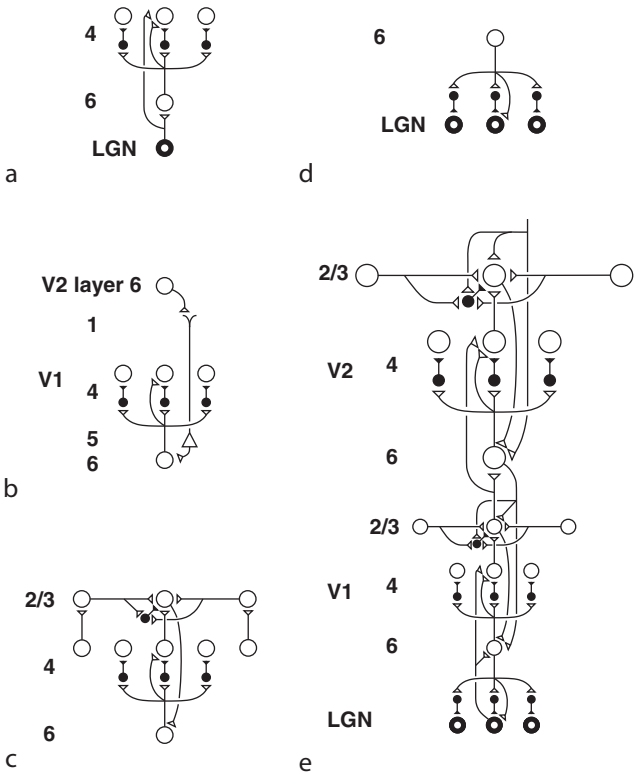
It has also been shown that the adaptive weights which are learned by some ART models can, at any stage of learning, be translated into fuzzy IF-THEN rules (Carpenter et al., 1992). Thus the ART model is a self-organizing rule-discovering production system as well as a neural network. These examples show that the claims of some cognitive scientists and AI practitioners that neural network models cannot learn rule-based behaviors are as incorrect as the claims that neural models cannot learn symbols.

How the Laminar Circuits of Cerebral Cortex Embody ART Mechanisms

More recent versions of ART have shown how predicted ART mechanisms may be embodied within known laminar microcircuits of the cerebral cortex. These include the family of LAMINART models (Fig. 2; see Raizada & Grossberg, 2003) and the synchronous matching ART, or SMART, model (Fig. 3; see Grossberg & Versace, 2008). SMART, in particular, predicts how a top-down match may lead to fast gamma oscillations that facilitate spike-timing dependent plasticity (STDP), whereas a mismatch can lead to slower beta oscillations that prevent learning by a STDP learning law. At least three neurophysiological labs have recently reported data consistent with the SMART prediction.

Review of ART and ARTMAP Algorithms

From Winner-Take-All to Distributed Coding As noted above, ART networks serve both as models of human cognitive information processing (Carpenter, 1997;



Adaptive Resonance Theory. Figure 2. LAMINART circuit clarifies how known cortical connections within and across cortical layers join the layer 6 → 4 and layer 2/3 circuits to form a laminar circuit model for the interblobs and pale stripe regions of cortical areas V1 and V2. Inhibitory interneurons are shown filled-in black. (a) The LGN provides bottom-up activation to layer 4 via two routes. First, it makes a strong connection directly into layer 4. Second, LGN axons send collaterals into layer 6, and thereby also activate layer 4 via the 6 → 4 on-center off-surround path. The combined effect of the bottom-up LGN pathways is to stimulate layer 4 via an on-center off-surround, which provides divisive contrast normalization (Grossberg, 1980) of layer 4 cell responses. (b)Folded feedback carries attentional signals from higher cortex into layer 4 of V1, via the modulatory 6 → 4 path. Corticocortical feedback axons tend preferentially to originate in layer 6 of the higher area and to terminate in layer 1 of the lower cortex, where they can excite the apical dendrites of layer 5 pyramidal cells whose axons send collaterals into layer 6. The triangle in the figure represents such a layer 5 pyramidal cell. Several other routes through which feedback can pass into V1 layer 6 exist. Having arrived in layer 6, the feedback is then "folded" back up into the feedforward stream by passing through the 6 → 4 on-center off-surround path (Bullier, Hup'e, James, & Girard, 1996). (c)Connecting the 6 → 4 on-centeroff-surround to the layer 2/3 grouping circuit: like-oriented layer 4 simple cells with opposite contrast polarities compete (not shown) before generating half-wave rectified outputs that converge onto layer 2/3 complex cells in the column above them. Just like attentional signals from higher cortex, as shown in (b), groupings that form within layer 2/3 also send activation into the folded feedback path, to enhance their own positions in layer 4 beneath them via the 6 → 4 on-center, and to suppress input to other groupings via the 6 → 4 off-surround. There exist direct layer 2/3 → 6 connections in macaque V1, as well as indirect routes via layer 5. (d) Top-down corticogeniculate feedback from V1 layer 6 to LGN also has an on-center off-surround anatomy, similar to the 6 → 4 path. The on-center feedback selectively enhances LGN cells that are consistent with the activation that they cause (Sillito, Jones, Gerstein, & West, 1994), and the off-surround contributes to length-sensitive (endstopped) responses that facilitate grouping perpendicular to line ends. (e) The entire V1/V2 circuit: V2 repeats the laminar pattern of V1 circuitry, but at a larger spatial scale. In particular, the horizontal layer 2/3 connections have a longer range in V2, allowing above-threshold perceptual groupings between more widely spaced

Grossberg, 1999, 2003) and as neural systems for technology transfer (Caudell, Smith, Escobedo, & Anderson, 1994; Parsons & Carpenter, 2003). Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of many variants of the basic networks, including fuzzy ARTMAP (Carpenter et al., 1992), ART-EMAP, ARTMAP-IC, and Gaussian ARTMAP. Early ARTMAP systems, including fuzzy ARTMAP, employ *winner-take-all (WTA) coding*, whereby each input activates a single category node during both training and testing. When a node is first activated during training, it is mapped to its designated output class.

Starting with ART-EMAP, subsequent systems have used *distributed coding* during testing, which typically improves predictive accuracy, while avoiding the computational problems inherent in the use of distributed code representations during training. In order to address these problems, distributed ARTMAP (Carpenter, 1997; Carpenter, Milenova, & Noeske, 1998) introduced a new network configuration, in addition to new learning laws.

Comparative analysis of the performance of ARTMAP systems on a variety of benchmark problems has led to the identification of a *default ARTMAP* network, which features simplicity of design and robust performance in many application domains. Default ARTMAP employs winner-take-all coding during training and distributed coding during testing within a distributed ARTMAP network architecture. With winner-take-all coding during testing, default ARTMAP reduces to a version of fuzzy ARTMAP.

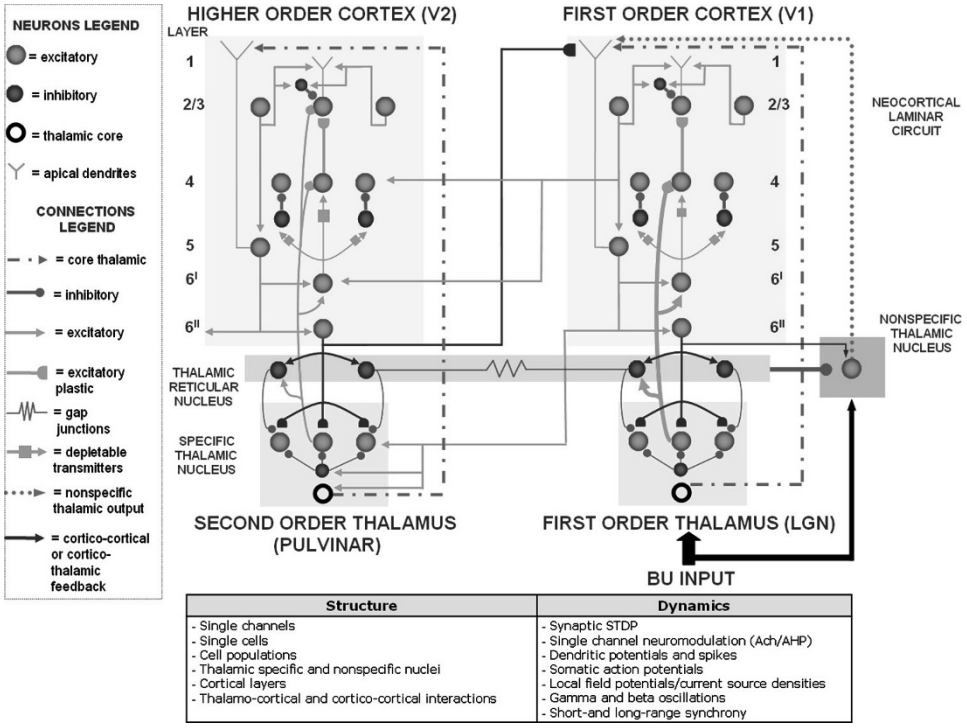
Complement Coding: Learning both Absent and Present Features ART and ARTMAP employ a preprocessing step called *complement coding* (Fig. 4), which models the nervous system's ubiquitous use of the computational design known as *opponent processing*. Balancing an entity against its opponent, as in agonist–antagonist muscle pairs, allows a system to act upon relative quantities, even as absolute magnitudes may vary unpredictably. In ART systems, complement coding is analogous to retinal ON-cells and OFF-cells. When the learning system is presented with a set of input features $\mathbf{a} \equiv (a_1 ... a_i ... a_M)$, complement coding doubles the number of input components, presenting to the network both the original feature vector and its complement.

Complement coding allows an ART system to encode within its critical feature patterns of memory features that are consistently *absent* on an equal basis with features that are consistently *present*. Features that are sometimes absent and sometimes present when a given category is learning are regarded as uninformative with respect to that category. Since its introduction, complement coding has been a standard element of ART and ARTMAP networks, where it plays multiple computational roles, including input normalization. However, this device is not particular to ART, and could, in principle, be used to preprocess the inputs to any type of system.

To implement complement coding, component activities a_i of a feature vector $\mathbf{a}$ are scaled; thus, $0 \leq a_i \leq 1$. For each feature i, the ON activity a_i determines the complementary OFF activity $(1 - a_i)$. Both a_i and $(1 - a_i)$ are represented in the $2M$-dimensional system input vector $\mathbf{A} = (\mathbf{a} \mid \mathbf{a}^c)$ (Fig. 4). Subsequent network computations then operate in this $2M$-dimensional input space. In particular, learned weight vectors $\mathbf{w}_J$ are $2M$-dimensional.

ARTMAP Search and Match Tracking in Fuzzy ARTMAP As illustrated by Fig. 1, the ART matching process triggers either learning or a parallel memory search. If search ends at an established code, the memory

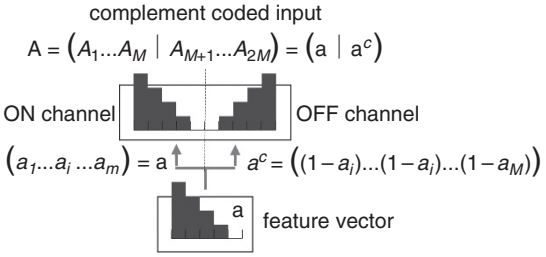
inducing stimuli to form. V1 layer 2/3 projects up to V2 layers 6 and 4, just as LGN projects to layers 6 an 4 of V1. Higher cortical areas send feedback into V2 which ultimately reaches layer 6, just as V2 feedback acts on layer 6 of V1. Feedback paths from higher cortical areas straight into V1 (not shown) can complement and enhance feedback from V2 into V1. Top-down attention can also modulate layer 2/3 pyramidal cells directly by activating both the pyramidal cells and inhibitory interneurons in that layer. The inhibition tends to balance the excitation, leading to a modulatory effect. These top-down attentional pathways tend to synapse in layer 1, as shown in Fig. 2b. Their synapses on apical dendrites in layer 1 are not shown, for simplicity. (Reprinted with permission from Raizada & Grossberg (2003))



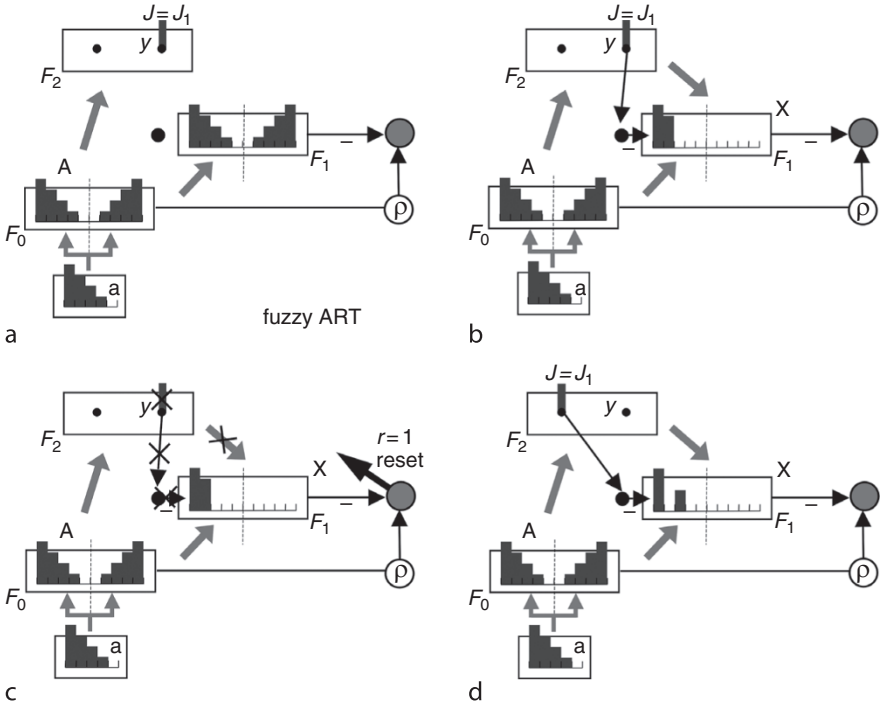
Adaptive Resonance Theory. Figure 3. SMART model overview. A first-order and higher-order cortical area are linked by corticocortical and corticothalamocortical connections. The thalamus is subdivided into specific first-order, second-order, nonspecific, and thalamic reticular nucleus (TRN). The thalamic matrix (one cell population shown as an open ring) provides priming to layer 1, where layer 5 pyramidal cell apical dendrites terminate. The specific thalamus relays sensory information (first-order thalamus) or lower-order cortical information (second-order thalamus) to the respective cortical areas via plastic connections. The nonspecific thalamic nucleus receives convergent BU input and inhibition from the TRN, and projects to layer 1 of the laminar cortical circuit, where it regulates reset and search in the cortical circuit (see text). Corticocortical feedback connections link layer 6^{II} of the higher cortical area to layer 1 of the lower cortical area, whereas thalamocortical feedback originates in layer 6^{II} and terminates in the specific thalamus after synapsing on the TRN. Layer 6^{II} corticothalamic feedback matches the BU input in the specific thalamus. V1 receives two parallel BU thalamocortical pathways. The LGN→V1 layer 4 pathway and the modulatory LGN→V1 layer 6^{I}→4 pathway provide divisive contrast normalization of layer 4 cell responses. The intracortical loop V1 layer 4→2/3→5→6^{I}→4 pathway (*folded feedback)* enhances the activity of winning layer 2/3 cells at their own positions via the 6^{I}→4 on-center, and suppresses input to other layer 2/3 cells via the 6^{I}→4 off-surround. V1 also activates the BU V1→V2 corticocortical pathways (V1 layer 2/3→V2 layers 6^{I} and 4) and the BU corticothalamocortical pathways (V1 layer 5→PULV→V2 layers 6^{I} and 4), where the layer 6^{I}→4 pathway provides divisive contrast normalization to V2 layer 4 cells analogously to V1. Corticocortical feedback from V2 layer 6^{II}→V1 layer 5→6^{I}→4 also uses the same modulatory 6^{I}→4 pathway. TRN cells of the two thalamic sectors are linked via gap junctions, which provide synchronization of the two thalamocortical sectors when processing BU stimuli (reprinted with permission from Grossberg & Versace (2008))

representation may either remain the same or incorporate new information from matched portions of the current input. While this dynamic applies to arbitrarily distributed activation patterns, the F_2 search and code for fuzzy ARTMAP (Fig. 5) describes a winner-take all system.

Before ARTMAP makes a class prediction, the bottom-up input **A** is matched against the top-down



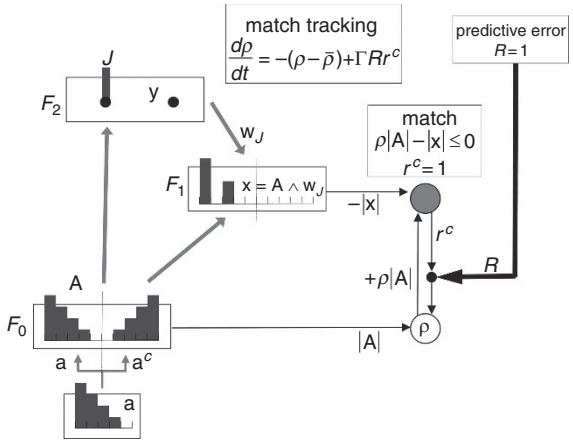
Adaptive Resonance Theory. Figure 4. Complement coding transforms an *M*-dimensional feature vector a into a 2*M*-dimensional system input vector A. A complement-coded system input represents both the degree to which a feature *i* is present (a_i) and the degree to which that feature is absent $(1 - a_i)$



Adaptive Resonance Theory. Figure 5. A fuzzy ART search cycle, with a distributed ART network configuration (Carpenter, 1997). The ART 1 search cycle (Carpenter and Grossberg, 1987) is the same, but allows only binary inputs and did not originally feature complement coding. The match field F_1 represents the matched activation pattern $\mathbf{x} = \mathbf{A} \wedge \mathbf{w}_J$, where $\wedge$ denotes the component-wise minimum, or fuzzy intersection, between the bottom-up input $\mathbf{A}$ and the top-down expectation $\mathbf{w}_J$. If the matched pattern fails to meet the matching criterion, then the active code is reset at F_2, and the system searches for another code y that better represents the input. The match/mismatch decision takes place in the ART orienting system. Each active feature in the input pattern $\mathbf{A}$ excites the orienting system with gain equal to the vigilance parameter ρ. Hence, with complement coding, the total excitatory input is $\rho|\mathbf{A}| = \rho\sum_{i=1}^{2M}\mathbf{A}_i = \rho M$. Active cells in the matched pattern x inhibit the orienting system, leading to a total inhibitory input equal to $-|\mathbf{x}| = -\sum_{i=1}^{2M} x_i$. If $\rho|\mathbf{A}| - |\mathbf{x}| \leq 0$, then the orienting system remains quiet, allowing resonance and learning to occur. If $\rho|\mathbf{A}| - |\mathbf{x}| > 0$, then the reset signal $r = 1$, initiating search for a better matching code

learned expectation, or critical feature pattern, that is read out by the active node (Fig. 5b). The matching criterion is set by a *vigilance* parameter ρ. As noted above, low vigilance permits the learning of abstract, prototype-like patterns, while high vigilance requires the learning of specific, exemplar-like patterns. When a new input arrives, vigilance equals a baseline level $\bar{\rho}$. Baseline vigilance is set equal to zero by default, in order to maximize generalization. Vigilance rises only after the system has made a predictive error. The internal control process that determines how far it must rise in order to correct the error is called *match tracking*. As vigilance rises, the network is required to pay more attention to how well top-down expectations match the current bottom-up input.

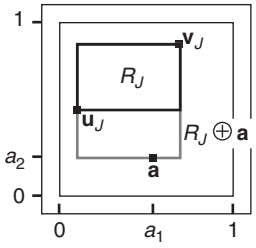
Match tracking (Fig. 6) forces an ARTMAP system not only to reset its mistakes, but to learn from them. With match tracking and fast learning, each ARTMAP network passes the *next input test*, which requires that, if a training input were re-presented immediately after a learning trial, it would directly activate the correct output class, with no predictive errors or search. Match tracking thus simultaneously implements the design goals of maximizing generalization and minimizing predictive error, without requiring the choice of a fixed matching criterion. ARTMAP memories thereby include both broad and specific pattern classes, with the latter typically formed as exceptions to the more general "rules" defined by the former. ARTMAP learning typically produces a wide variety of such mixtures, whose exact composition depends upon the order of training exemplar presentation.



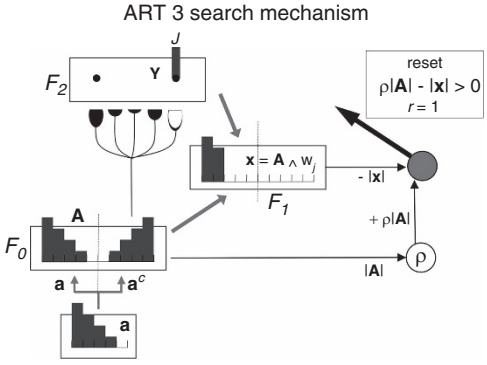
Adaptive Resonance Theory. Figure 6. ARTMAP match tracking. When an active node *J* meets the matching criterion ($\rho\,|\mathbf{A}| - |\mathbf{x}| \le 0$), the reset signal $r = 0$ and the node makes an prediction. If the predicted output is incorrect, the feedback signal $R = 1$. While $R = r^c = 1$, r increases rapidly. As soon as $\rho > \frac{|\mathbf{x}|}{|\mathbf{A}|}$, *r* switches to 1, which both halts the increase of r and resets the active F_2 node. From one chosen node to the next, *r* decays to slightly below $\frac{|\mathbf{x}|}{|\mathbf{A}|}$ (MT–). On the time scale of learning *r* returns to $\bar{\rho}$

Unless they have already activated all their coding nodes, ARTMAP systems contain a reserve of nodes that have never been activated, with weights at their initial values. These *uncommitted* nodes compete with the previously active *committed* nodes, and an uncommitted node is chosen over poorly matched committed nodes. An ARTMAP design constraint specifies that an active uncommitted node should not reset itself. Weights initially begin with $w_{iJ} = 1$. Thus, when the active node J is uncommitted, $\mathbf{x} = \mathbf{A} \wedge \mathbf{w}_J = \mathbf{A}$ at the match field. Then, $\rho\,|\mathbf{A}| - |\mathbf{x}| = \rho\,|\mathbf{A}| - |\mathbf{A}| = (\rho - 1)\,|\mathbf{A}|$. Thus $\rho\,|\mathbf{A}| - |\mathbf{x}| \le 0$ and an uncommitted node does not trigger a reset, provided $\rho \le 1$.

ART Geometry Fuzzy ART long-term memories are visualized as hyper-rectangles, called *category boxes*. The weight vector $\mathbf{w}_J$ is interpreted geometrically as a box R_J whose ON-channel corner $\mathbf{u}_J$ and OFF-channel corner $\mathbf{v}_J$ are, in the format of the complement-coded input vector, defined by $\left(\mathbf{u}_J \,\middle|\, \mathbf{v}_J^C\right) \equiv \mathbf{w}_J$ (Fig. 7). For fuzzy ART with the choice-by-difference $F_0 \to F_2$ signal function T_J, an input $\mathbf{a}$ activates the node J of the closest category box R_J, according to the L_1 (city-block) metric. In case of a tie, as when $\mathbf{a}$ lies in more than one box, the node with the smallest R_J is chosen, where $|R_J|$ is defined as the sum of the edge lengths $\sum_{i=1}^{M} |v_{iJ} - u_{iJ}|$. The chosen node J will reset if $|R_J \oplus \mathbf{a}| > M\,(1 - \rho)$, where $R_J \oplus \mathbf{a}$ is the smallest box enclosing both R_J and $\mathbf{a}$. Otherwise, R_J expands toward $R_J \oplus \mathbf{a}$ during learning. With fast learning, $R_J^{\text{new}} = R_J^{\text{old}} \oplus \mathbf{a}$.



Adaptive Resonance Theory. Figure 7. Fuzzy ART geometry. The weight of a category node *J* is represented in complement-coding form as $\mathbf{w}_J = (\mathbf{u}_J \mid \mathbf{v}_J^c)$, and the *M*-dimensional vectors $\mathbf{u}_J$ and $\mathbf{v}_J$ define the corners of the category box R_J. When $M = 2$, the size of R_J equals its width plus its height. During learning, R_J expands toward $R_J \oplus \mathbf{a}$, defined as the smallest box enclosing both R_J and a. Node *J* will reset before learning if $|R_J \oplus \mathbf{a}| > M(1 - \rho)$



Adaptive Resonance Theory. Figure 8. ART 3 search implements a medium-term memory within the F_0-to-F_2 pathways, which biases the system against choosing a category node that had just produced a reset

Biasing Against Previously Active Category Nodes and Previously Attended Features During Attentive Memory Search Activity $\mathbf{x}$ at the ART field F_1 continuously computes the match between the field's bottom-up and top-down input patterns. A reset signal r shuts off the active F_2 node J when $\mathbf{x}$ fails to meet the matching criterion determined by the value of the vigilance parameter ρ. Reset alone does not, however, trigger a search for a different F_2 node: unless the prior activation has left an enduring trace within the F_0-to-F_2 subsystem, the network will simply reactivate the same node as before. As modeled in ART 3, biasing the bottom-up input to the coding field F_2 to favor the previously inactive nodes implements search by allowing the network to activate a new node in response to a reset signal. The ART 3 search mechanism defines a medium-term memory (MTM) in the F_0-to-F_2 adaptive filter which biases the system against re-choosing a node that had just produced a reset. A presynaptic interpretation of this bias is transmitter depletion, or habituation (Fig. 8).

Medium-term memory in all ART models allows the network to shift attention among learned categories at the coding field F_2 during search. The new *biased ART* network (Carpenter & Gaddam, 2010) introduces a second medium-term memory that shifts attention among input features, as well as categories, during search.

Self-Organizing Rule Discovery This foundation of computational principles and mechanisms has enabled the development of an ART information fusion system that is capable of incrementally learning a cognitive hierarchy of rules in response to probabilistic, incomplete, and even contradictory data that are collected by multiple observers (Carpenter, Martens, & Ogas, 2005).

Cross References

►Bayes Rule
►Bayesian Methods

Recommended Reading

Bullier, J., Hupé, J. M., James, A., & Girard, P. (1996). Functional interactions between areas V1 and V2 in the monkey. *Journal of Physiology Paris*, *90*(3–4), 217–220.

Carpenter, G. A. (1997). Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. *Neural Networks*, *10*, 1473–1494.

Carpenter, G. A. & Gaddam, S. C. (2010). Biased ART: A neural architecture that shifts attention towards previously disregarded features following an incorrect prediction. *Neural Networks*, 23.

Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37, 54–115.

Carpenter, G. A. & Grossberg, S. (1993). Normal and amnesic learning, recognition, and memory by a neural model of cortico-hippocampal interactions. *Trends in Neurosciences*, *16*, 131–137.

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H. & Rosen, D. B. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, *3*, 698–713.

Carpenter, G. A., Martens, S., & Ogas, O. J. (2005). Self-organizing information fusion and hierarchical knowledge discovery: A

new framework using ARTMAP neural networks. *Neural Networks*, *18*, 287–295.
Carpenter, G. A., Milenova, B. L., & Noeske, B. W. (1998). Distributed ARTMAP: A neural network for fast distributed supervised learning. *Neural Networks*, *11*, 793–813.
Caudell, T. P., Smith, S. D. G., Escobedo, R., & Anderson, M. (1994). NIRS: Large scale ART 1 neural architectures for engineering design retrieval. *Neural Networks*, *7*, 1339–1350.
Grossberg, S. (1976). Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, *23*, 187–202.
Grossberg, S. (1980). How does a brain build a cognitive code? *Psychological Review*, *87*, 1–51.
Grossberg, S. (1999). The link between brain, learning, attention, and consciousness. *Consciousness and Cognition*, *8*, 1–44.
Grossberg, S. (2000). The complementary brain: Unifying brain dynamics and modularity. *Trends in Cognitive Sciences*, *4*, 233–246.
Grossberg, S. (2003). How does the cerebral cortex work? Development, learning, attention, and 3D vision by laminar circuits of visual cortex. *Behavioral and Cognitive Neuroscience Reviews, 2*, 47–76.
Grossberg, S. (2007). Consciousness CLEARS the mind. *Neural Networks*, *20*, 1040–1053.
Grossberg, S. & Versace, M. (2008). Spikes, synchrony, and attentive learning by laminar thalamocortical circuits. *Brain Research*, *1218*, 278–312.
Parsons, O., & Carpenter, G. A. (2003). ARTMAP neural networks for information fusion and data mining: Map production and target recognition methodologies. *Neural Networks*, *16*(7), 1075–1089.
Posner, M. I., & Keele, S. W. (1968). On the genesis of abstract ideas. *Journal of Experimental Psychology*, *77*, 353–363.
Raizada, R., & Grossberg, S. (2003). Towards a theory of the laminar architecture of cerebral cortex: Computational clues from the visual system. *Cerebral Cortex, 13*, 100–113.
Sillito, A. M., Jones, H. E., Gerstein, G. L., & West, D. C. (1994). Feature-linked synchronization of thalamic relay cell firing induced by feedback from the visual cortex. *Nature*, *369*, 479–482.

Adaptive System

►Complexity in Adaptive Systems

Agent

In computer science, the term "agent" usually denotes a software abstraction of a real entity which is capable of acting with a certain degree of autonomy. For example, in artificial societies, agents are software abstractions of real people, interacting in an artifical, simulated environment. Various authors have proposed different definitions of agents. Most of them would agree on the following set of agent properties:

- Persistence: Code is not executed on demand but runs continuously and decides autonomously when it should perform some activity.
- Social ability: Agents are able to interact with other agents.
- Reactivity: Agents perceive the environment and are able to react.
- Proactivity: Agents exhibit goal-directed behavior and can take the initiative.

Agent-Based Computational Models

►Artificial Societies

Agent-Based Modeling and Simulation

►Artificial Societies

Agent-Based Simulation Models

►Artificial Societies

AIS

►Artificial Immune Systems

Algorithm Evaluation

Geoffrey I. Webb
Monash University, Victoria, Australia

Definition

Algorithm evaluation is the process of assessing a property or properties of an algorithm.

Motivation and Background

It is often valuable to assess the efficacy of an algorithm. In many cases, such assessment is relative, that is,

evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many learning algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to infer properties of the algorithm, such as its computational complexity (Papadimitriou, 1994), and also employs the tools of ►computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such as time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a ►model, that is, at ►training time, and performance when applying a learned model, that is, at ►test time.

Other types of property that are often studied are the properties of the models that are learned (see ►model evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance on any application from relative performance on this sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's ►bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than directly being properties of the models that are learned. Hence, they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias–variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include ►bootstrap sampling, ►cross-validation, and ►holdout evaluation.

Cross References

►Computational Learning Theory
►Model Evaluation

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. New York: Springer.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.
Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison-Wesley.
Webb, G. I. (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, *40*(2), 159–196.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Analogical Reasoning

►Instance-Based Learning

Analysis of Text

►Text Mining

Analytical Learning

►Deductive Learning
►Explanation-Based Learning

Ant Colony Optimization

Marco Dorigo, Mauro Birattari
Université Libre de Bruxelles, Brussels, Belgium

Synonyms

ACO

Definition

Ant colony optimization (ACO) is a population-based metaheuristic for the solution of difficult combinatorial

optimization problems. In ACO, each individual of the population is an artificial agent that builds incrementally and stochastically a solution to the considered problem. Agents build solutions by moving on a graph-based representation of the problem. At each step their moves define which solution components are added to the solution under construction. A probabilistic model is associated with the graph and is used to bias the agents' choices. The probabilistic model is updated online by the agents so as to increase the probability that future agents will build good solutions.

Motivation and Background

Ant colony optimization is so called because of its original inspiration: the foraging behavior of some ant species. In particular, in Beckers, Deneubourg, and Goss (1992) it was demonstrated experimentally that ants are able to find the shortest path between their nest and a food source by collectively exploiting the pheromone they deposit on the ground while walking. Similar to real ants, ACO's artificial agents, also called artificial ants, deposit artificial pheromone on the graph of the problem they are solving. The amount of pheromone each artificial ant deposits is proportional to the quality of the solution the artificial ant has built. These artificial pheromones are used to implement a probabilistic model that is exploited by the artificial ants to make decisions during their solution construction activity.

Structure of the Optimization System

Let us consider a minimization problem $(\mathcal{S}, f)$, where $\mathcal{S}$ is the *set of feasible solutions*, and f is the *objective function*, which assigns to each solution $s \in \mathcal{S}$ a cost value $f(s)$. The goal is to find an optimal solution s^*, that is, a feasible solution of minimum cost. The set of all optimal solutions is denoted by $\mathcal{S}^*$.

Ant colony optimization attempts to solve this minimization problem by repeating the following two steps:

- Candidate solutions are constructed using a parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is intended to bias future sampling toward low cost solutions.

The Ant Colony Optimization Probabilistic Model

We assume that the combinatorial optimization problem $(\mathcal{S}, f)$ is mapped on a problem that can be characterized by the following list of items:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_C}\}$ of *components*, where N_C is the number of components.
- A finite set $\mathcal{X}$ of *states* of the problem, where a state is a sequence $x = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ over the elements of $\mathcal{C}$. The length of a sequence x, that is, the number of components in the sequence, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < +\infty$.
- A set of (candidate) solutions $\mathcal{S}$, which is a subset of $\mathcal{X}$ (i.e., $\mathcal{S} \subseteq \mathcal{X}$).
- A set of feasible states $\tilde{\mathcal{X}}$, with $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, defined via a set of *constraints* Ω.
- A nonempty set $\mathcal{S}^*$ of optimal solutions, with $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ and $\mathcal{S}^* \subseteq \mathcal{S}$.

Given the above formulation (Note that, because this formulation is always possible, ACO can in principle be applied to any combinatorial optimization problem.) artificial ants build candidate solutions by performing randomized walks on the completely connected, weighted graph $\mathcal{G} = (\mathcal{C}, \mathcal{L}, \mathcal{T})$, where the vertices are the components $\mathcal{C}$, the set $\mathcal{L}$ fully connects the components $\mathcal{C}$, and $\mathcal{T}$ is a vector of so-called *pheromone trails* τ. Pheromone trails can be associated with components, connections, or both. Here we assume that the pheromone trails are associated with connections, so that $\tau(i,j)$ is the pheromone associated with the connection between components i and j. It is straightforward to extend the algorithm to the other cases. The graph $\mathcal{G}$ is called the *construction graph*.

To construct candidate solutions, each artificial ant is first put on a randomly chosen vertex of the graph. It then performs a randomized walk by moving at each step from vertex to vertex on the graph in such a way that the next vertex is chosen stochastically according to the strength of the pheromone currently on the arcs.

While moving from one node to another of the graph $\mathcal{G}$, constraints Ω may be used to prevent ants from building infeasible solutions. Formally, the solution construction behavior of a generic ant can be described as follows:

ANT_SOLUTION_CONSTRUCTION

- For each ant:
  - Select a start node c_1 according to some problem dependent criterion.
  - Set $k = 1$ and $x_k = \langle c_1 \rangle$.
- While $x_k = \langle c_1, c_2, \ldots, c_k \rangle \in \tilde{\mathcal{X}}, x_k \notin \mathcal{S}$, and the set J_{x_k} of components that can be appended to x_k is not empty, select the next node (component) c_{k+1} randomly according to:

$$P_{\mathcal{T}}(c_{k+1} = c | x_k) = \begin{cases} \dfrac{F_{(c_k,c)}(\tau(c_k,c))}{\sum_{(c_k,y)\in J_{x_k}} F_{(c_k,y)}(\tau(c_k,y))} & \text{if } (c_k,c) \in J_{x_k}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where a connection (c_k, y) belongs to J_{x_k} if and only if the sequence $x_{k+1} = \langle c_1, c_2, \ldots, c_k, y \rangle$ satisfies the constraints Ω (that is, $x_{k+1} \in \tilde{\mathcal{X}}$) and $F_{(i,j)}(z)$ is some monotonic function – a common choice being $z^{\alpha}\eta(i,j)^{\beta}$, where $\alpha, \beta > 0$, and $\eta(i,j)$'s are heuristic values measuring the desirability of adding component j after i. If at some stage $x_k \notin \mathcal{S}$ and $J_{x_k} = \emptyset$, that is, the construction process has reached a dead-end, the current state x_k is discarded. However, this situation may be prevented by allowing artificial ants to build infeasible solutions as well. In such a case, an infeasibility penalty term is usually added to the cost function. Nevertheless, in most of the settings in which ACO has been applied, the dead-end situation does not occur.

For certain problems, one may find it useful to use a more general scheme, where F depends on the pheromone values of several "related" connections rather than just a single one. Moreover, instead of the *random-proportional rule* above, different selection schemes, such as the *pseudo-random-proportional rule* (Dorigo & Gambardella, 1997), may be used.

The Ant Colony Optimization Pheromone Update

Many different schemes for pheromone update have been proposed within the ACO framework. For an extensive overview, see Dorigo and Stützle (2004). Most pheromone updates can be described using the following generic scheme:

GENERIC_ACO_UPDATE

- $\forall s \in \hat{S}_t, \forall (i,j) \in s: \ \tau(i,j) \leftarrow \tau(i,j) + Q_f(s|S_1, \ldots, S_t)$,
- $\forall (i,j): \ \tau(i,j) \leftarrow (1-\rho) \cdot \tau(i,j)$,

where S_i is the sample in the ith iteration, ρ, $0 \le \rho < 1$, is the evaporation rate, and $Q_f(s|S_1, \ldots, S_t)$ is some "quality function," which is typically required to be non-increasing with respect to f and is defined over the "reference set" $\hat{S}_t$.

Different ACO algorithms may use different quality functions and reference sets. For example, in the very first ACO algorithm – Ant System (Dorigo, Maniezzo, & Colorni, 1991, 1996) – the quality function is simply $1/f(s)$ and the reference set $\hat{S}_t = S_t$. In a subsequently proposed scheme, called *iteration best update* (Dorigo & Gambardella, 1997), the reference set is a singleton containing the best solution within S_t (if there are several iteration-best solutions, one of them is chosen randomly). For the *global-best update* (Dorigo et al., 1996; Stützle & Hoos, 1997), the reference set contains the best among all the iteration-best solutions (and if there are more than one global-best solution, the earliest one is chosen). In Dorigo et al. (1996) an *elitist* strategy was introduced, in which the update is a combination of the previous two.

In case a good lower bound on the optimal solution cost is available, one may use the following quality function (Maniezzo, 1999):

$$Q_f(s|S_1, \ldots, S_t) = \tau_0 \left(1 - \frac{f(s) - \mathrm{LB}}{\bar{f} - \mathrm{LB}}\right) = \tau_0 \frac{\bar{f} - f(s)}{\bar{f} - \mathrm{LB}}, \tag{2}$$

where $\bar{f}$ is the average of the costs of the last k solutions and LB is the lower bound on the optimal solution cost. With this quality function, the solutions are evaluated by comparing their cost to the average cost of the other recent solutions, rather than by using the absolute cost values. In addition, the quality function is automatically scaled based on the proximity of the average cost to the lower bound.

A pheromone update that slightly differs from the generic update described above was used in *ant colony system* (ACS) (Dorigo & Gambardella, 1997). There the pheromone is evaporated by the ants online during the solution construction, hence only the pheromone involved in the construction evaporates.

Another modification of the generic update was introduced in $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (Stützle & Hoos, 1997, 2000), which uses maximum and minimum pheromone trail limits. With this modification, the probability of generating any particular solution is kept above some positive threshold. This helps to prevent search stagnation and premature convergence to suboptimal solutions.

Cross References

►Swarm Intelligence

Recommended Reading

Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology, 159*, 397–415.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Dorigo M., Maniezzo V., & Colorni A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26*(1), 29–41.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT Press.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing, 11*(4), 358–369.

Stützle, T., & Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In *Proceedings of the 1997 Congress on Evolutionary Computation – CEC'97* (pp. 309–314). Piscataway, NJ: IEEE Press.

Stützle, T., & Hoos, H. H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems, 16*(8), 889–914, 2000.

Anytime Algorithm

An *anytime algorithm* is an algorithm whose output increases in quality gradually with increased running time. This is in contrast to algorithms that produce no output at all until they produce full-quality output after a sufficiently long execution time. An example of an algorithm with good anytime performance is ►Adaptive Real-Time Dynamic Programming (ARTDP).

AODE

►Averaged One-Dependence Estimators

Apprenticeship Learning

►Behavioral Cloning

Approximate Dynamic Programming

►Value Function Approximation

Apriori Algorithm

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996) is a ►data mining method which outputs all ►frequent itemsets and ►association rules from given data.

Input: set $\mathcal{I}$ of items, multiset $\mathcal{D}$ of subsets of $\mathcal{I}$, frequency threshold *min_fr*, and confidence threshold *min_conf*.

Output: all frequent itemsets and all valid association rules in $\mathcal{D}$.

Method:

1: level := 1; frequent_sets := $\emptyset$;
2: candidate_sets := $\{\{i\} \mid i \in \mathcal{I}\}$;
3: while candidate_sets $\neq \emptyset$
3.1: scan data $\mathcal{D}$ to compute frequencies of all sets in candidate_sets;
3.2: frequent_sets := frequent_sets $\cup$ $\{C \in$ candidate_sets $\mid$ frequency$(C) \geq$ *min_fr*$\}$;
3.3 level := level + 1;
3.4: candidate_sets := $\{A \subset \mathcal{I} \mid |A| =$ level and $B \in$ frequent_sets for all $B \subset A, |B| =$ level $- 1\}$;

4: output frequent_sets;
5: for each $F \in$ frequent_sets
5.1: for each $E \subset F, E \neq \emptyset, E \neq F$
5.1.1: if frequency(F)/frequency$(E) \geq$ *min_conf* then output association rule $E \rightarrow (F \setminus E)$

The algorithm finds frequent itemsets (lines 1-4) by a breadth-first, general-to-specific search. It generates and tests candidate itemsets in batches, to reduce the overhead of database access. The search starts with the most general itemset patterns, the singletons, as candidate patterns (line 2). The algorithm then iteratively computes the frequencies of candidates (line 3.1) and saves those that are frequent (line 3.2). The crux of the algorithm is in the candidate generation (line 3.4): on the next level, those itemsets are pruned that have an infrequent subset. Obviously, such itemsets cannot be frequent. This allows Apriori to find all frequent itemset without spending too much time on infrequent itemsets. See ►frequent pattern and ►constraint-based mining for more details and extensions.

Finally, the algorithm tests all frequent association rules and outputs those that are also confident (lines 5-5.1.1).

Cross References

►Association Rule
►Basket Analysis
►Constraint-Based Mining
►Frequent Itemset
►Frequent Pattern

Recommended Reading

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–s328). Menlo Park: AAAI Press.

Area Under Curve

Synonyms

AUC

Definition

The *area under curve* (AUC) statistic is an empirical measure of classification performance based on the area under an ROC curve. It evaluates the performance of a scoring classifier on a test set, but ignores the magnitude of the scores and only takes their rank order into account. AUC is expressed on a scale of 0 to 1, where 0 means that all negatives are ranked before all positives, and 1 means that all positives are ranked before all negatives. See ►ROC Analysis.

AQ

►Rule Learning

ARL

►Average-Reward Reinforcement Learning

ART

►Adaptive Resonance Theory

ARTDP

►Adaptive Real-Time Dynamic Programming

Artificial Immune Systems

Jon Timmis
University of York, Heslington, North Yorkshire, UK

Synonyms

AIS; Immune computing; Immune-inspired computing; Immunocomputing; Immunological computation

Definition

Artificial immune systems (AIS) have emerged as a computational intelligence approach that shows great promise. Inspired by the complexity of the immune system, computer scientists and engineers have created systems that in some way mimic or capture certain computationally appealing properties of the immune system, with the aim of building more robust and adaptable solutions. AIS have been defined by de Castro and Timmis (2002) as:

- adaptive systems, inspired by theoretical immunology and observed immune functions, principle and models, which are applied to problem solving

AIS are not limited to machine learning systems, there are a wide variety of other areas in which AIS are developed such as optimization, scheduling, fault tolerance, and robotics (Hart & Timmis, 2008). Within the context of machine learning, both supervised and unsupervised approaches have been developed. Immune-inspired learning approaches typically develop a memory set of detectors that are capable of classifying unseen data items (in the case of supervised learning) or a memory set of detectors that represent clusters within the data (in the case of unsupervised learning). Both static and dynamic learning systems have been developed.

Motivation and Background

The immune system is a complex system that undertakes a myriad of tasks. The abilities of the immune system have helped to inspire computer scientists to build systems that *mimic*, in some way, various properties of the immune system. This field of research, AIS, has seen the application of immune-inspired algorithms to a wide variety of areas.

The origin of AIS has its roots in the early theoretical immunology work of Farmer, Perelson, and Varela (Farmer, Packard, & Perelson, 1986; Varela, Coutinho, Dupire, & Vaz, 1988). These works investigated a number of theoretical ▶immune network models proposed to describe the maintenance of immune memory in the absence of antigen. While controversial from an immunological perspective, these models began to give rise to an interest from the computing community. The most influential people at crossing the divide between computing and immunology in the early days were Bersini and Forrest. It is fair to say that some of the early work by Bersini (1991) was very well rooted in immunology, and this is also true of the early work by Forrest (1994). It was these works that formed the basis of a solid foundation for the area of AIS. In the case of Bersini, he concentrated on the immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicking that property. With regard to Forrest, her work was focused on computer security (in particular, network intrusion detection) and formed the basis of a great deal of further research by the community on the application of immune-inspired techniques to computer security.

At about the same time as Forrest was undertaking her work, other researchers began to investigate the nature of learning in the immune system and how that might by used to create *machine learning* algorithms (Cook & Hunt, 1995). They had the idea that it might be possible to exploit the mechanisms of the immune system (in particular, the immune network) in learning systems, so they set about doing a proof of concept (Cook & Hunt, 1995). Initial results were very encouraging, and they built on their success by applying the immune ideas to the classification of DNA sequences as either promoter or nonpromoter classes: this work was generalized in Timmis and Neal (2001).

Similar work was carried out by de Castro and Von Zuben (2001), who developed algorithms for use in function optimization and data clustering. Work in dynamic unsupervised machine learning algorithms was also undertaken, meeting with success in works such as Neal (2002). In the supervised learning domain, very little happened until the work by Watkins (2001) (later expanded in Watkins, 2005) developed an immune-based classifier known as AIRS, and in the dynamic supervised domain, with the work in Secker, Freitas, and Timmis (2003) being one of a number of successes.

Structure of the Learning System

In an attempt to create a common basis for AIS, the work in de Castro and Timmis (2002) proposed the idea of a framework for engineering AIS. They argued that the case for such a framework as the existence of similar frameworks in other biologically inspired approaches, such as ▶artificial neural networks (ANNs) and evolutionary algorithms (EAs), has helped considerably with the understanding and construction of such systems. For example, de Castro and Timmis (2002) consider a set of artificial neurons, which can be arranged together to form an ANN. In order to acquire knowledge, these neural networks undergo an adaptive process, known as learning or training, which alters (some of) the parameters within the network. Therefore, they argued that in a simplified form, a framework to design an ANN is

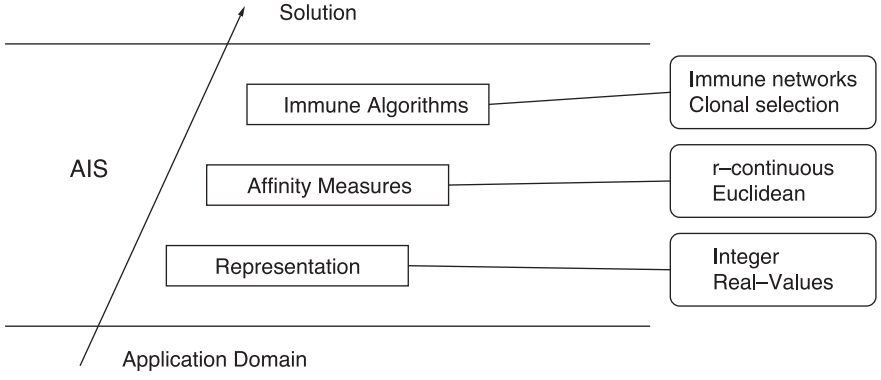
composed of a set of artificial neurons, a pattern of interconnection for these neurons, and a learning algorithm. Similarly, they argued that in evolutionary algorithms, there is a set of artificial chromosomes representing a population of individuals that iteratively suffer a process of reproduction, genetic variation, and selection. As a result of this process, a population of evolved artificial individuals arises. A framework, in this case, would correspond to the genetic representation of the individuals of the population, plus the procedures for reproduction, genetic variation, and selection. Therefore, they proposed that a framework to design a biologically inspired algorithm requires, at least, the following basic elements:

- A representation for the components of the system
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other. The environment is usually stimulated by a set of input stimuli, one or more fitness function(s), or other means
- Procedures of adaptation that govern the dynamics of the system, i.e., how its behavior varies over time

This framework can be thought of as a layered approach such as the specific framework for engineering AIS of de Castro and Timmis (2002) shown in Fig. 1. This framework follows the three basic elements for designing a biologically inspired algorithm just described, where the set of mechanisms for evaluation are the affinity measures and the procedures of adaptation are the immune algorithms. In order to build a system such as an AIS, one typically requires an application domain or target function. From this basis, the way in which the components of the system will be represented is considered. For example, the representation of network traffic may well be different from the representation of a real-time embedded system. In AIS, the way in which something is represented is known as *shape space*. There are many kinds of shape space, such as Hamming, real valued, and so on, each of which carries it own bias and should be selected with care (Freitas & Timmis, 2003). Once the representation has been chosen, one or more affinity measures are used to quantify the interactions of the elements of the system. There are many possible affinity measures (which are partially dependent upon the representation adopted), such as Hamming and Euclidean distance metrics. Again, each of these has its own bias, and the affinity function must be selected with great care, as it can affect the overall performance (and ultimately the result) of the system (Freitas & Timmis, 2003).

Supervised Immune-Inspired Learning

The artificial immune recognition system (AIRS) algorithm was introduced as one of the first immune-inspired supervised learning algorithms and has subsequently gone through a period of study and refinement (Watkins, 2005). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, AIRS is a, ▶clonal selection type of immune-inspired algorithm. The representation and affinity layers of the system are standard in



Artificial Immune Systems. Figure 1. AIS layered framework adapted from de Castro and Timmis (2002)

that any number of representations such as binary, real values, etc., can be used with the appropriate affinity function. AIRS has its origin in two other immune-inspired algorithms: CLONALG (CLONAL Selection alGorithm) and Artificial Immune NEtwork (AINE) (de Castro and Timmis, 2002). AIRS resembles CLONALG in the sense that both the algorithms are concerned with developing a set of memory cells that give a representation of the learned environment.

AIRS is concerned with the development of a set of memory cells that can encapsulate the training data. This is done in a two-stage process of first evolving a candidate memory cell and then determining if this candidate cell should be added to the overall pool of memory cells. The learning process can be outlined as follows:

1. For each pattern to be recognized, do
   (a) Compare a training instance with all memory cells of the same class and find the memory cell with the best affinity for the training instance. This is referred to as a memory cell mc_{match}.
   (b) Clone and mutate mc_{match} in proportion to its affinity to create a pool of abstract B-cells.
   (c) Calculate the affinity of each B-cell with the training instance.
   (d) Allocate resources to each B-cell based on its affinity.
   (e) Remove the weakest B-cells until the number of resources returns to a preset limit.
   (f) If the average affinity of the surviving B-cells is above a certain level, continue to step 1(g). Else, clone and mutate these surviving B-cells based on their affinity and return to step 1(c).
   (g) Choose the best B-cell as a candidate memory cell (mc_{cand}).
   (h) If the affinity of mc_{cand} for the training instance is better than the affinity of mc_{match}, then add mc_{cand} to the memory cell pool. If, in addition to this, the affinity between mc_{cand} and mc_{match} is within a certain threshold, then remove mc_{match} from the memory cell pool.
2. Repeat from step 1(a) until all training instances have been presented.

Once this training routine is complete, AIRS classifies the instances using k-nearest neighbor with the developed set of memory cells.

Unsupervised Immune-Inspired Learning

The artificial immune network (aiNET) algorithm was introduced as one of the first immune-inspired unsupervised learning algorithms and has subsequently gone through a period of study and refinement (de Castro & Von Zuben, 2001). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, aiNET is an immune network type of immune-inspired algorithm. The representation and affinity layers of the system are standard (the same as in AIRS). aiNET has its origin in another immune-inspired algorithms: CLONALG (the same forerunner to AIRS), and resembles CLONALG in the sense that both algorithms (again) are concerned with developing a set of memory cells that give a representation of the learnt environment. However, within aiNET there is no error feedback into the learning process. The learning process can be outlined as follows:

1. Randomly initialize a population P
2. For each pattern to be recognized, do
   (a) Calculate the affinity of each B-cell (b) in the network for an instance of the pattern being learnt
   (b) Select a number of elements from P into a clonal pool C
   (c) Mutate each element of C proportional to affinity to the pattern being learnt (the higher the affinity, the less mutation applied)
   (d) Select the highest affinity members of C to remain in the set C and remove the remaining elements
   (e) Calculate the affinity between all members of C and remove elements in C that have an affinity below a certain threshold (user defined)
   (f) Combine the elements of C with the set P
   (g) Introduce a random number of randomly created elements into P to maintain diversity
3. Repeat from 2(a) until stopping criteria is met

Once this training routine is complete, the minimum-spanning tree algorithm is applied to the network to extract the clusters from within the network.

Recommended Reading

Bersini, H. (1991). Immune network and adaptive control. In *Proceedings of the 1st European conference on artificial life (ECAL)* (pp. 217–226). Cambridge, MA: MIT Press.

Cooke, D., & Hunt, J. (1995). Recognising promoter sequences using an artificial immune system. In *Proceedings of intelligent systems in molecular biology* (pp. 89–97). California: AAAI Press.

de Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. New York: Springer.

de Castro, L. N., & Von Zuben, F. J. (2001). *aiNet: An artificial immune network for data analysis* (pp. 231–259). Hershey, PA: Idea Group Publishing.

Farmer, J. D., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D, 22*, 187–204.

Forrest, S., Perelson, A. S., Allen, L., Cherukuri, R. (1994). Self–nonself discrimination in a computer. In *Proceedings of the IEEE symposium on research security and privacy* (pp. 202–212).

Freitas, A., & Timmis, J. (2003). *Revisiting the foundations of artificial immune systems: A problem oriented perspective*, LNCS (Vol. 2787) (pp. 229–241). New York: Springer.

Hart, E., & Timmis, J. (2008). Application Areas of AIS: The Past, Present and the Future. *Journal of Applied Soft Computing, 8*(1). pp. 191–201.

Neal, M. (2002). An artificial immune system for continuous analysis of time-varying data. In J. Timmis & P. Bentley (Eds.), *Proceedings of the 1st international conference on artificial immune system (ICARIS)* (pp. 76–85). Canterbury, UK: University of Kent Printing Unit.

Secker, A., Freitas, A., & Timmis, J. (2003). AISEC: An artificial immune system for email classification. In *Proceedings of congress on evolutionary computation (CEC)* (pp. 131–139).

Timmis, J., & Bentley (Eds.). (2002). *Proceedings of the 1st international conference on artificial immune system (ICARIS)*. Canterbury, UK: University of Kent Printing Unit.

Timmis, J., & Neal, M. (2001). A resource limited artificial immune system for data analysis. *Knowledge Based Systems, 14*(3–4), 121–130.

Varela, F., Coutinho, A., Dupire, B., & Vaz, N. (1988). Cognitive networks: Immune, neural and otherwise. *Journal of Theoretical Immunology, 2*, 359–375.

Watkins, A. (2001). AIRS: A resource limited artificial immune classifier. Master's thesis, Mississippi State University.

Watkins, A. (2005). Exploiting immunological metaphors in the development of serial, parallel and distributed learning algorithms. PhD thesis, University of Kent.

Artificial Life

Artificial Life is an interdisciplinary research area trying to reveal and understand the principles and organization of living systems. Its main goal is to artificially synthesize life-like behavior from scratch in computers or other artificial media. Important topics in artificial life include the origin of life, growth and development, evolutionary and ecological dynamics, adaptive autonomous robots, emergence and self-organization, social organization, and cultural evolution.

Artificial Neural Networks

(ANNs) is a computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

Cross References

►Adaptive Resonance Theory
►Backpropagation
►Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity
►Boltzmann Machines
►Cascade Correlation
►Competitive Learning
►Deep Belief Networks
►Evolving Neural Networks
►Hypothesis Language
►Neural Network Topology
►Neuroevolution
►Radial Basis Function Networks
►Reservoir Computing
►Self-Organizing Maps
►Simple Recurrent Networks
►Weights

Artificial Societies

Jürgen Branke
University of Warwick, Coventry, UK

Synonyms

Agent-based computational models; Agent-based modeling and simulation; Agent-based simulation models

Definition

An artificial society is an agent-based, computer-implemented simulation model of a society or group of people, usually restricted to their interaction in a particular situation. Artificial societies are used in economics and social sciences to explain, understand, and analyze socioeconomic phenomena. They provide scientists with a fully controllable virtual laboratory to test hypotheses and observe complex system behavior emerging as result of the ►agents' interaction. They allow formalizing and testing social theories by using computer code, and make it possible to use experimental methods with social phenomena, or at least with their computer representations, on a large scale. Because the designer is free to choose any desired ►agent behavior as long as it can be implemented, research based on artificial societies is not restricted by assumptions typical in classical economics, such as homogeneity and full rationality of agents. Overall, artificial societies have added an all new dimension to research in economics and social sciences and have resulted in a new research field called "agent-based computational economics."

Artificial societies should be distinguished from virtual worlds and ►artificial life. The term virtual world is usually used for virtual environments to interact with, as, e.g., in computer games. In artificial life, the goal is more to learn about biological principles, understand how life could emerge, and create life within a computer.

Motivation and Background

Classical economics can be roughly divided into analytical and empirical approaches. The former uses deduction to derive theorems from assumptions. Thereby, analytical models usually include a number of simplifying assumptions in order to keep the model tractable, the most typical being full rationality and homogeneity of agents. Also, analytical economics is often limited to equilibrium calculations. Classical empirical economics collects data from the real world, and derives patterns and regularities inductively. In recent years, the tremendous increase in available computational power gave rise to a new branch of economics and sociology which uses simulation of artificial societies as a tool to generate new insights.

Artificial societies are agent-based, computer-implemented simulation models of real societies or a group of people in a specific situation. They are built from the bottom up, by specifying the behavior of the agents in different situations. The simulation then reveals the emerging global behavior of the system, and thus provides a link between micro-level behavior of the agents and macro-level characteristics of the system. Using simulation, researchers can now carry out social experiments under fully controlled and reproducible laboratory conditions, trying out different configurations and observing the consequences.

Like deduction, simulation models are based on a set of clearly specified assumptions as written down in a computer program. This is then used to generate data, from which regularities and patterns are derived inductively. As such, research based on artificial societies stands somewhere between the classical analytical and empirical social sciences.

One of the main advantages of artificial societies is that they allow to consider very complex scenarios where agents are heterogeneous, boundedly rational, or have the ability to learn. Also, they allow to observe evolution over time, instead of just the equilibrium.

Artificial societies can be used for many purposes, e.g.:

1. Verification: Test a hypothesis or theory by examining its validity in relevant, clearly defined scenarios.
2. Explanation: Construct an artificial society which shows the same behavior as the real society. Then analyze the model to explain the emergent behavior.
3. Prediction: Run a model of an existing society into the future. Also, feed the model with different input parameters and use the result as a prediction on how the society would react.
4. Optimization: Test different strategies in the simulation environment, trying to find a best possible strategy.
5. Existence proof: Demonstrate that a specific simulation model is able to generate a certain global behavior.
6. Discovery: Play around with parameter settings, discovering new interdependencies and gaining new insights.
7. Training and education: Use simulation as demonstrator.

Structure of the Learning System

Using artificial societies requires the usual steps in model building and experimental science, including

1. Developing a conceptual model
2. Building the simulation model
3. Verification (making sure the model is correct)
4. Validation (making sure the model is suitable to answer the posed questions)
5. Simulation and analysis using an appropriate experimental design.

Artificial society is an interdisciplinary research area involving, among others, computer science, psychology, economics, sociology, and biology.

Important Aspects

The modeling, simulation, and analysis process described in the previous section is rather complex and only remotely connected to machine learning. Thus, instead of a detailed description of all steps, the following focuses on aspects particularly interesting from a machine learning point of view.

Modeling Learning

One of the main advantages of artificial societies is that they can account for boundedly rational and learning agents. For that, one has to specify (in form of a program) exactly how agents decide and learn.

In principle, all the learning algorithms developed in machine learning could be used, and many have been used successfully, including ►reinforcement learning, ►artificial neural networks, and ►evolutionary algorithms. However, note that the choice of a learning algorithm is not determined by its learning speed and efficiency (as usual in machine learning), but by how well it reflects human learning in the considered scenario, at least if the goal is to construct an artificial society which allows conclusions to be transferred to the real world. As a consequence, many learning models used in artificial societies are motivated by psychology. The idea of the most suitable model depends on the simulation context, e.g., on whether the simulated learning process is conscious or nonconscious, or on the time and effort an individual may be expected to spend on a particular decision.

Besides individual learning (i.e., learning from own past experience), artificial societies usually feature social learning (where one agent learns by observing others), and cultural learning (e.g., the evolution of norms). While the latter simply emerges from the interaction of the agents, the former has to be modeled explicitly. Several different models for learning in artificial societies are discussed in Brenner (2006).

One popular learning paradigm which can be used as a model for individual as well as social learning are ►evolutionary algorithms (EAs). Several studies suggest that EAs are indeed an appropriate model for learning in artificial societies, either based on comparisons of simulations with human subject experiments or based on comparisons with other learning mechanisms such as reinforcement learning (Duffy, 2006). As EAs are successful search strategies, they seem particularly suitable if the space of possible actions or strategies is very large.

If used to model individual learning, each agent uses a separate EA to search for a better personal solution. In this case, the EA population represents the different alternative actions or strategies that an agent considers. The genetic operators crossover and mutation are clearly related to two major ingredients of human innovation: combination and variation. Crossover can be seen as deriving a new concept by combining two known concepts, and mutation corresponds to a small variation of an existing concept. So, the agent, in some sense, creatively tries out new possibilities. Selection, which favors the best solutions found so far, models the learning part. A solution's quality is usually assessed by evaluating it in a simulation assuming all other agents keep their behavior.

For modeling social learning, EAs can be used in two different ways. In both cases, the population represents the actions or strategies of the different agents in the population. From this it follows that the population size corresponds to the number of agents in the simulation. Fitness values are calculated by running the simulation and observing how the different agents perform. Crossover is now seen as a model for information exchange, or imitation, among agents. Mutation, as in the individual learning case, is seen as a small variation of an existing concept.

The first social learning model simply uses a standard EA, i.e., selection chooses agents to "reproduce,"

and the resulting new agent strategy replaces an old strategy in the population. While allowing to use standard EA libraries, this approach does not provide a direct link between agents in the simulation and individuals in the EA population. In the second social learning model, each agent directly corresponds to an individual in the EA. In every iteration, each agent creates and tests a new strategy as follows. First, it selects a "donor" individual, with preference to successful individuals. Then it performs a crossover of its own strategy and the donor's strategy, and mutates the result. This can be regarded as an agent observing other agents, and partially adopting the strategies of successful other agents. Then, the resulting new strategy is tested in a "thought experiment," by testing whether the agent would be better off with the new strategy compared with its current strategy, assuming all other agents keep their behavior. If the new strategy performs better, it replaces the current strategy in the next iteration. Otherwise, the new strategy is discarded and the agent again uses its old strategy in the next iteration. The testing of new strategies against their parents has been termed election operator in Arifovic (1994), and makes sure that some very bad and obviously implausible agent strategies never enter the artificial society.

Examples

One of the first forerunners of artificial societies was Schelling's segregation model, 1969. In this study, Schelling placed some artificial agents of two different colors on a simple grid. Each agent follows a simple rule: if less than a given percentage of agents in the neighborhood had the same color, the agent moves to a random free spot. Otherwise, it stays. As the simulation shows, in this model, segregation of agent colors could be observed even if every individual agent was satisfied to live in a neighborhood with only 50% of its neighbors being of the same color. Thus, with this simple model, Schelling demonstrated that segregation of races in suburbs can occur even if each individual would be happy to live in a diverse neighborhood. Note that the simulations were actually not implemented on a computer but carried out by moving coins on a grid by hand.

Other milestones in artificial societies are certainly the work by Epstein and Axtell on their "sugarscape" model (Epstein & Axtell, 1996), and the Santa Fe artificial stock market (Arthur, Holland, LeBaron, Palmer, & Taylor, 1997). In the former, agents populate a simple grid world, with sugar growing as the only resource. The agents need the sugar for survival, and can move around to collect it. Axtell and Epstein have shown that even with agents following some very simple rules, the emerging behavior of the overall system can be quite complex and similar in many aspects to observations in the real world, e.g., showing a similar wealth distribution or population trajectories.

The latter is a simple model of a stock market with only a single stock and a risk-free fixed-interest alternative. This model has subsequently been refined and studied by many researchers. One remarkable result of the first model was to demonstrate that technical trading can actually be a viable strategy, something widely accepted in practice, but which classical analytical economics struggled to explain.

One of the most sophisticated artificial societies is perhaps the model of the Anasazi tribe, who left their dwellings in the Long House Valley in northeastern Arizona for so far unknown reasons around 1300 BC (Axtell et al., 2002). By building an artificial society of this tribe and the natural surroundings (climate etc.), it was possible to replicate macro behavior which is known to have occurred and provide a possible explanation for the sudden move.

The NewTies project (Gilbert et al., 2006) has a different and quite ambitious focus: it constructs artificial societies with the hope of an emerging artificial language and culture, which then might be studied to help explain how language and culture formed in human societies.

Software Systems

Agent-based simulations can be facilitated by using specialized software libraries such as Ascape, Netlogo, Repast, StarLogo, Mason, and Swarm. A comparison of different libraries can be found in Railsback, Lytinen, and Jackson (2006).

Applications

Artificial societies have many practical applications, from rather simple simulation models to very complex economic decision problems, examples include

traffic simulation, market design, evaluation of vaccination programs, evacuation plans, or supermarket layout optimization. See, e.g., Bonabeau (2002) for a discussion of several such applications.

Future Directions, Challenges

The science on artificial societies is still at its infancy, but the field is burgeoning and has already produced some remarkable results. Major challenges lie in the model building, calibration, and validation of the artificial society simulation model. Despite several agent-based modeling toolkits available, there is a lot to be gained by making them more flexible, intuitive, and user-friendly, allowing to construct complex models simply by selecting and combining provided building blocks of agent behavior. ►Behavioral Cloning may be a suitable machine learning approach to generate representative agent models.

Cross References

►Artificial Life
►Behavioral Cloning
►Co-Evolutionary Learning
►Multi-Agent Learning

Recommended Reading

Agent-based computational economics, website maintained by Tesfatsion (2009)

Axelrod: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration (Axelrod, 1997)

Bonabeau: Agent-based modeling (Bonabeau, 2002)

Brenner: Agent learning representation: Advice on modeling economic learning (Brenner, 2006)

Epstein: Generative social science (Epstein, 2006)

Journal of Artificial Societies and Social Simulation (2009)

Tesfatsion and Judd (eds.): Handbook of computational economics (Tesfatsion & Judd, 2006)

Arifovic, J. (1994). Genetic algorithm learning and the cobweb-model. *Journal of Economic Dynamics and Control, 18*, 3–28.

Arthur, B., Holland, J., LeBaron, B., Palmer, R., & Taylor, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In B. Arthur et al., (Eds.), *The economy as an evolvin complex system II* (pp. 15–44). Boston: Addison-Wesley.

Axelrod, R. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton, NJ: Princeton University Press.

Axtell, R. L., Epstein, J. M., Dean, J. S., Gumerman, G. J., Swedlund, A. C., Harburger, J., et al. (2002). Population growth and collapse in a multiagent model of the kayenta anasazi in long house valley. *Proceedings of the National Academy of Sciences, 99*, 7275–7279.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, *99*, 7280–7287.

Brenner, T. (2006). Agent learning representation: Advice on modelling economic learning. In L. Tesfatsion & K. L. Judd, (Eds.), *Handbook of computational economics* (Vol. 2, pp.895–947). Amsterdam: North-Holland.

Duffy, J. (2006). Agent-based models and human subject experiments. In L. Tesfatsion & K. L. Judd, (Eds.), *Handbook of computational economics* (Vol. 2, pp.949–1011). Amsterdam: North-Holland.

Epstein, J. M. (2006). *Generative social science: Studies in agent-based computational modeling*. Princeton, NJ: Princeton University Press.

Epstein, J. M., & Axtell, R. (1996). *Growing artificial societies*. Washington, DC: Brookings Institution Press.

Gilbert, N., den Besten, M., Bontovics, A., Craenen, B. G. W., Divina, F., Eiben, A. E., et al. (2006). Emerging artificial societies through learning. *Journal of Artificial Societies and Social Simulation, 9*(2). http://jasss.soc.surrey.ac.uk/9/2/9.html.

Railsback, S. F., Lytinen, S. L., & Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation, 82*(9), 609–623.

Schelling, T. C. (1969). Dynamic models of segregation. *Journal of Mathematical Sociology, 2*, 143–186.

Tesfatsion, L. (2009). Website on agent-based computational economics. http://www.econ.iastate.edu/tesfatsi/ace.htm.

Tesfatsion, L., & Judd, K. L. (Eds.) (2006). *Handbook of computational economics – Vol 2: Agent-based computational economics*. Amsterdam: Elsevier.

The journal of artificial societies and social simulation. http://jasss.soc.surrey.ac.uk/JASSS.html.

Assertion

In ►Minimum Message Length, the code or language shared between sender and receiver that is used to describe the model.

Association Rule

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Association rules (Agrawal, Imieliński, & Swami, 1993) can be extracted from data sets where each example

consists of a set of items. An association rule has the form $X \rightarrow Y$, where X and Y are ►itemsets, and the interpretation is that if set X occurs in an example, then set Y is also likely to occur in the example.

Each association rule is usually associated with two statistics measured from the given data set. The *frequency* or *support* of a rule $X \rightarrow Y$, denoted $\mathrm{fr}(X \rightarrow Y)$, is the number (or alternatively the relative frequency) of examples in which $X \cup Y$ occurs. Its *confidence*, in turn, is the observed conditional probability $P(Y \mid X) = \mathrm{fr}(X \cup Y)/\mathrm{fr}(X)$.

The ►Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen & Verkamo, 1996) finds all association rules, between any sets X and Y, which exceed user-specified support and confidence thresholds. In association rule mining, unlike in most other learning tasks, the result thus is a set of rules concerning different subsets of the feature space.

Association rules were originally motivated by supermarket ►basket analysis, but as a domain independent technique they have found applications in numerous fields. Association rule mining is part of the larger field of ►frequent itemset or ►frequent pattern mining.

Cross References

►Apriori Algorithm
►Basket Analysis
►Frequent Itemset
►Frequent Pattern

Recommended Reading

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: AAAI Press.

Associative Bandit Problem

►Associative Reinforcement Learning

Associative Reinforcement Learning

Alexander L. Strehl
Rütgers University, USA

Synonyms

Associative bandit problem; Bandit problem with side information; Bandit problem with side observations; One-step reinforcement learning

Definition

The *associative reinforcement-learning* problem is a specific instance of the ►*reinforcement learning* problem whose solution requires *generalization* and *exploration* but not *temporal credit assignment*. In associative reinforcement learning, an action (also called an arm) must be chosen from a fixed set of actions during successive timesteps and from this choice a real-valued reward or payoff results. On each timestep, an input vector is provided that along with the action determines, often probabilistically, the reward. The goal is to maximize the expected long-term reward over a finite or infinite horizon. It is typically assumed that the action choices do not affect the sequence of input vectors. However, even if this assumption is not asserted, learning algorithms are not required to infer or model the relationship between input vectors from one timestep to the next. Requiring a learning algorithm to discover and reason about this underlying process results in the full reinforcement learning problem.

Motivation and Background

The problem of associative reinforcement learning may be viewed as connecting the problems of ►supervised learning or ►classification, which is more specific, and reinforcement learning, which is more general. Its study is motivated by real-world applications such as choosing which internet advertisements to display based on information about the user or choosing which stock to buy based on current information related to the market. Both problems are distinguished from supervised learning by the absence of labeled training examples to learn from. For instance, in the advertisement problem, the learner is never told which ads would have resulted in the greatest expected reward (in this problem, reward is

determined by whether an ad is clicked on or not). In the stock problem, the best choice is never revealed since the choice itself affects the future price of the stocks and therefore the payoff.

The Learning Setting

The learning problem consists of the following core objects:

- An input space $\mathcal{X}$, which is a set of objects (often a subset of the n-dimension Euclidean space $\mathbb{R}^n$).
- A set of actions or arms $\mathcal{A}$, which is often a finite set of size k.
- A distribution D over $\mathcal{X}$. In some cases, D is allowed to be time-dependent and may be denoted D_t on timestep t for $t = 1, 2, \ldots$.

A learning sequence proceeds as follows. During each timestep $t = 1, 2, \ldots$, an input vector $x_t \in \mathcal{X}$ is is drawn according to the distribution D and is provided to the algorithm. The algorithm selects an aarm at $a_t \in \mathcal{A}$. This choice may be stochastic and depend on all previous inputs and rewards observed by the algorithm as well as all previous action choices made by the algorithm for timesteps $t = 1, 2, \ldots$. Then, the learner receives a payoff r_t generated according to some unknown stochastic process that depends only on the x_t and a_t. The informal goal is to maximize the expected long-term payoff. Let $\pi : \mathcal{X} \to \mathcal{A}$ be any policy that maps input vectors to actions. Let

$$V^{\pi}(T) := E\left[\sum_{i=1}^{T} r_i \,\middle|\, a_i = \pi(x_i) \text{ for } i = 1, 2, \ldots, T\right] \quad (1)$$

denotes the expected total reward over T steps obtained by choosing arms according to policy π. The expectation is taken over any randomness in the generation of input vectors x_i and rewards r_i. The expected regret of a learning algorithm with respect to policy π is defined as $V^{\pi}(T) - E[\sum_{i=1}^{T} r_i]$ the expected difference between the return from following policy π and the actual obtained return.

Power of Side Information

Wang, Kulkarni, and Poor (2005) studied the associative reinforcement learning problem from a statistical viewpoint. They considered the setting with two action and analyzed the *expected inferior sampling time*, which is the number of times that the lesser action, in terms of expected reward, is selected. The function mapping input vectors to conditional reward distributions belongs to a known parameterized class of functions, with the true parameters being unknown. They show that, under some mild conditions, an algorithm can achieve finite expected inferior sampling time. This demonstrates the power provided by the input vectors (also called *side observations* or *side information*), because such a result is not possible in the standard *multi-armed bandit problem*, which corresponds to the associative reinforcement-learning problem without input vectors x_i. Intuitively, this type of result is possible when the side information can be used to infer the payoff function of the optimal action.

Linear Payoff Functions

In its most general setting, the associative reinforcement learning problem is intractable. One way to rectify this problem is to assume that the payoff function is described by a linear system. For instance, Abe and Long (1999) and Auer (2002) consider a model where during each timestep t, there is a vector $z_{t,i}$ associated with each arm i. The expected payoff of pulling arm i on this timestep is given by $\theta^T z_{t,i}$ where θ is an unknown parameter vector and θ^T denotes the transpose of f. This framework maps to the framework described above by taking $x_t = (z_{t,1}, z_{t,2}, \ldots, z_{t,k})$. They assume a time-dependent distribution D and focus on obtaining bounds on the regret against the optimal policy. Assuming that all rewards lie in the interval $[0, 1]$, the worst possible regret of any learning algorithm is linear. When considering only the number of timesteps T, Auer (2002) shows that a regret (with respect to the optimal policy) of $O(\sqrt{T}\ln(T))$ can be obtained.

PAC Associative Reinforcement Learning

The previously mentioned works analyze the growth rate of the regret of a learning algorithm with respect to the optimal policy. Another way to approach the problem is to allow the learner some number of timesteps of *exploration*. After the exploration trials, the algorithm is required to output a policy. More specifically, given inputs $0 < \epsilon < 1$ and $0 < \delta < 1$, the algorithm is

required to output an ϵ-optimal policy with probability at least $1 - \delta$. This type of analysis is based on the work by Valiant (1984), and learning algorithms satisfying the above condition are termed *probably approximately correct* (PAC).

Motivated by the work of Kaelbling (1994), Fiechter (1995) developed a PAC algorithm when the true payoff function can be described by a *decision list* over the action and input vector. Building on both works, Strehl, Mesterharm, Littman, and Hirsh (2006) showed that a class of associative reinforcement learning problems can be solved efficiently, in a PAC sense, when given a learning algorithm for efficiently solving classification problems.

Recommended Reading

Section 6.1 of the survey by Kaelbling, Littman, and Moore (1996) presents a nice overview of several techniques for the associative reinforcement-learning problem, such as CRBP (Ackley and Littman, 1990), ARC (Sutton, 1984), and REINFORCE (Williams, 1992).

Abe, N., & Long, P. M. (1999). Associative reinforcement learning using linear probabilistic concepts. In *Proceedings of the 16th international conference on machine learning* (pp. 3–11).

Ackley, D. H., & Littman, M. L. (1990). Generalization and scaling in reinforcement learning. In *Advances in neural information processing systems 2* (pp. 550–557). San Mateo, CA: Morgan Kaufmann.

Auer, P. (2002). Using confidence bounds for exploitation–exploration trade-offs. *Journal of Machine Learning Research, 3*, 397–422.

Fiechter, C.-N. (1995). PAC associative reinforcement learning. Unpublished manuscript.

Kaelbling, L. P. (1994). Associative reinforcement learning: Functions in *k*-DNF. *Machine Learning, 15*, 279–298.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Strehl, A. L., Mesterharm, C., Littman, M. L., & Hirsh, H. (2006). Experience-efficient learning in associative bandit problems. In *ICML-06: Proceedings of the 23rd international conference on machine learning*, Pittsburgh, Pennsylvania (pp. 889–896).

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. Doctoral dissertation, University of Massachusetts, Amherst, MA.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.

Wang, C.-C., Kulkarni, S. R., & Poor, H. V. (2005). Bandit problems with side observations. *IEEE Transactions on Automatic Control, 50*, 3988–3993.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*, 229–256.

Attribute

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Characteristic; Feature; Property; Trait

Definition

Attributes are properties of things, ways that we, as humans, might describe them. If we were talking about the appearance of our friends, we might describe one of them as "sex female," "hair brown," "height 5 ft 7 in." Linguistically, this is rather terse, but this very terseness has the advantage of limiting ambiguity. The attributes are sex, hair color, and height. For each friend, we could give the appropriate values to go along with each attribute, some examples are shown in Table 1. Attribute-value pairs are a standard way of describing things within the machine learning community. Traditionally, values have come in one of three types: binary, sex has two values; nominal, hair color has many values; real, height has an ordered set of values. Ideally, the attribute-value pairs are sufficient to describe some things accurately and to tell them apart from others. What might be described is very varied, so the attributes themselves will vary widely.

Motivation and Background

For machine learning to be successful, we need a language to describe everyday things that is sufficiently powerful to capture the similarities and differences between them and yet is computationally easy to manage. The idea that a sufficient number of attribute-value

Attribute. Table 1 Some friends

Sex	Hair color	Height
Male	Black	6 ft 2 in.
Female	Brown	5 ft 7 in.
Female	Blond	5 ft 9 in.
Male	Brown	5 ft 10 in.

pairs would meet this requirement is an intuitive one. It has also been studied extensively in philosophy and psychology, as a way that humans represent things mentally. In the early days of artificial intelligence research, the frame (Minsky, 1974) became a common way of representing knowledge. We have, in many ways, inherited this representation, attribute-value pairs sharing much in common with the labeled slots for values used in frames. In addition, the data for many practical problems comes in this form. Popular methods of storing and manipulating data such as relational databases, and less formal structures such as spread sheets, have columns as attributes and cells as values. So, attribute-value pairs are a ubiquitous way of representing data.

Future Directions

The notion of an attribute-value pair is so well entrenched in machine learning that it is difficult to perceive what might replace it. As, in many practical applications, the data comes in this form, this representation will undoubtedly be around for some time. One change that is occurring is the growing complexity of attribute-values. Traditionally, we have used the simple value types, binary, nominal, and real, discussed earlier. But to effectively describe many things, we need to extend this simple language and use more complex values. For example, in ►data mining applied to multimedia, more new complex representations abound. Sound and video streams, images, and various properties of them, are just a few examples (Cord et al., 2005; Simoff & Djeraba, 2000).

Perhaps, the most significant change is away from attributes, albeit with complex values, to structural forms where the relationship between things is included. As Quinlan (1996) states "Data may concern objects or observations with arbitrarily complex structure that cannot be captured by the values of a predetermined set of attributes." There is a large and growing community of researchers in ►relational learning. This is evidenced by the number, and growing frequency, of recent workshops at the International Conference for Machine Learning (Cord et al., 2005; De Raedt & Kramer, 2000; Dietterich, Getoor, & Murphy, 2004; Fern, Getoor, & Milch, 2006).

Limitations

In philosophy there is the idea of essence, the properties an object must have to be what it is. In machine learning, particularly in practical applications, we get what we are given and have little control in the choice of attributes and their range of values. If domain experts have chosen the attributes, we might hope that they are properties that can be readily ascertained and are relevant to the task at the hand. For example, when describing one of our friends, we would not say Fred is the one with the spleen. It is not only difficult to observe, it is also poor at discriminating between people. Data are collected for many reasons. In medical applications, all sorts of attribute-values would be collected on patients. Most are unlikely to be important to the current task. An important part of learning is ►feature extraction, determining which attributes are necessary for learning.

Whether or not attribute-value pairs are an essential representation for the type of learning required in the development, and functioning, of intelligent agents, remains to be seen. Attribute-values readily capture symbolic information, typically at the level of words that humans naturally use. But if our agents need to move around in their environment, recognizing what they encounter, we might need a different nonlinguistic representation. Certainly, other representations based on a much finer granularity of features, and more holistic in nature, have been central to areas such as ►neural networks for some time. In research into ►dynamic systems, attractors in a sensor space might be more realistic that attribute-values (See chapter on ►Classification).

Recommended Reading

Cord, M., Dahyot, R., Cunningham, P., & Sziranyi, T. (Eds.). (2005). Workshop on machine learning techniques for processing multimedia content. In *Proceedings of the twenty-second international conference on machine learning*.

De Raedt, L., & Kramer, S. (Eds.). (2000). In *Proceedings of the seventeenth international conference on machine learning*. Workshop on attribute-value and relational learning: Crossing the boundaries, Stanford University, Palo Alto, CA.

Dietterich, T., Getoor, L., & Murphy, K. (Eds.). (2004). In *Proceedings of the twenty-first international conference on machine learning*. Workshop on statistical relational learning and its connections to other fields.

Fern, A., Getoor, L., & Milch, B. (Eds.). (2006). In *Proceedings of the twenty-fourth international conference on machine learning*. Workshop on open problems in statistical relational learning.

Minsky, M. (1974). A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA.

Quinlan, J. R. (1996). Learning first-order definitions of functions. *Journal of Artificial Intelligence Research, 5*, 139–161.

Simoff, S. J., & Djeraba, C. (Eds.). (2000). In *Proceedings of the sixth international conference on knowledge discovery and data mining*. Workshop on multimedia data mining.

Attribute Selection

►Feature Selection

Attribute-Value Learning

Attribute-value learning refers to any learning task in which the each ►Instance is described by the values of some finite set of attributes (see ►Attribute). Each of these instances is often represented as a vector of attribute values, each position in the vector corresponding to a unique attribute.

AUC

►Area Under Curve

Autonomous Helicopter Flight Using Reinforcement Learning

Adam Coates[1], Pieter Abbeel[2], Andrew Y. Ng[3]
[1]Stanford University, Stanford, CA, USA
[2]University of California, Berkeley, CA, USA
[3]Stanford University, Stanford, CA, USA

Definition

Helicopter flight is a highly challenging control problem. While it is possible to obtain controllers for simple maneuvers (like hovering) by traditional manual design procedures, this approach is tedious and typically requires many hours of adjustments and flight testing, even for an experienced control engineer. For complex maneuvers, such as aerobatic routines, this approach is likely infeasible. In contrast, ►reinforcement learning (RL) algorithms enable faster and more automated design of controllers. Model-based RL algorithms have been used successfully for autonomous helicopter flight for hovering, forward flight and, using apprenticeship learning methods for expert-level aerobatics. In model-based RL, first one builds a model of the helicopter dynamics and specifies the task using a reward function. Then, given the model and the reward function, the RL algorithm finds a controller that maximizes the expected sum of rewards accumulated over time.

Motivation and Background

Autonomous helicopter flight represents a challenging control problem and is widely regarded as being significantly harder than control of fixed-wing aircraft. (See, e.g., Leishman, (2000); Seddon, (1990)). At the same time, helicopters provide unique capabilities such as in-place hover, vertical takeoff and landing, and low-speed maneuvering. These capabilities make helicopter control an important research problem for many practical applications.

Building autonomous flight controllers for helicopters, however, is far from trivial. When done by hand, it can require many hours of tuning by experts with extensive prior knowledge about helicopter dynamics. Meanwhile, the automated development of helicopter controllers has been a major success story for RL methods. Controllers built using RL algorithms have established state-of-the-art performance for both basic flight maneuvers, such as hovering and forward flight (Bagnell & Schneider, 2001; Ng, Kim, Jordan, & Sastry, 2004), as well as being among the only successful methods for advanced aerobatic stunts. Autonomous helicopter aerobatics has been successfully tackled using the innovation of "apprenticeship learning," where the algorithm learns by watching a human demonstrator (Abbeel & Ng, 2004). These methods have enabled autonomous helicopters to fly aerobatics as well as an expert human pilot, and often even better (Coates, Abbeel, & Ng, 2008).

Developing autonomous flight controllers for helicopters is challenging for a number of reasons:

1. Helicopters have unstable, high-dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics. As a consequence, all successful helicopter flight

controllers (to date) have many parameters. Controllers with 10–100 gains are not atypical. Hand engineering the right setting for each of the parameters is difficult and time consuming, especially since their effects on performance are often highly coupled through the helicopter's complicated dynamics. Moreover, the unstable dynamics, especially in the low-speed flight regime, complicates flight testing.

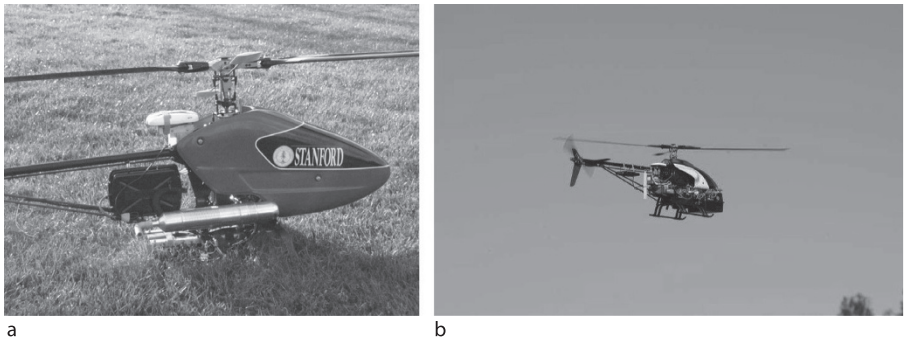
2. Helicopters are underactuated: their position and orientation is representable using six parameters, but they have only four control inputs. Thus helicopter control requires significant planning and making trade-offs between errors in orientation and errors in desired position.
3. Helicopters have highly complex dynamics: Even though we describe the helicopter as having a twelve dimensional state (position, velocity, orientation, and angular velocity), the true dynamics are significantly more complicated. To determine the precise effects of the inputs, one would have to consider the airflow in a large volume around the helicopter, as well as the parasitic coupling between the different inputs, the engine performance, and the non-rigidity of the rotor blades. Highly accurate simulators are thus difficult to create, and controllers developed in simulation must be sufficiently robust that they generalize to the real helicopter in spite of the simulator's imperfections.
4. Sensing capabilities are often poor: For small remotely controlled (RC) helicopters, sensing is limited because the on-board sensors must deal with a large amount of vibration caused by the helicopter blades rotating at about 30 Hz, as well as higher frequency noise from the engine. Although noise at these frequencies (which are well above the roughly 10 Hz at which the helicopter dynamics can be modeled reasonably) might be easily removed by low pass filtering, this introduces latency and damping effects that are detrimental to control performance. As a consequence, helicopter flight controllers have to be robust to noise and/or latency in the state estimates to work well in practice.

Typical Hardware Setup

A typical autonomous helicopter has several basic sensors on board. An Inertial Measurement Unit (IMU) measures angular rates and linear accelerations for each of the helicopter's three axes. A 3-axis magnetometer senses the direction of the Earth's magnetic field, similar to a magnetic compass (Fig. 1).

Attitude-only sensing, as provided by the inertial and magnetic sensors, is insufficient for precise, stable hovering, and slow-speed maneuvers. These maneuvers require that the helicopter maintain relatively tight control over its position error, and hence high-quality position sensing is needed. GPS is often used to determine helicopter position (with carrier-phase GPS units achieving sub-decimeter accuracy), but vision-based solutions have also been employed (Abbeel, Coates, Quigley, & Ng, 2007; Coates et al., 2008; Saripalli, Montgomery, & Sukhatme, 2003).

Vibration adds errors to the sensor measurements and may damage the sensors themselves, hence significant effort may be required to mount the sensors on the airframe (Dunbabin, Brosnan, Roberts, & Corke, 2004). Provided there is no aliasing, sensor errors added by



a b

Autonomous Helicopter Flight Using Reinforcement Learning. Figure 1. (a) Stanford University's instrumented XCell Tempest autonomous helicopter. (b) A Bergen Industrial Twin autonomous helicopter with sensors and on-board computer

vibration can be removed by using a digital filter on the measurements (though, again, one must be careful to avoid adding too much latency).

Sensor data from the aircraft sensors is used to estimate the state of the helicopter for use by the control algorithm. This is usually done with an extended Kalman filter (EKF). A unimodal distribution (as computed by the EKF) suffices to represent the uncertainty in the state estimates and it is common practice to use the mode of the distribution as the state estimate for feedback control. In general the accuracy obtained with this method is sufficiently high that one can treat the state as fully observed.

Most autonomous helicopters have an on-board computer that runs the EKF and the control algorithm (Gavrilets, Martinos, Mettler, & Feron, 2002a; La Civita, Papageorgiou, Messner, & Kanade, 2006; Ng et al., 2004). However, it is also possible to use ground-based computers by sending sensor data by wireless to the ground, and then transmitting control signals back to the helicopter through the pilot's RC transmitter (Abbeel et al., 2007; Coates et al., 2008).

Helicopter State and Controls

The helicopter state s is defined by its position (p_x, p_y, p_z), orientation (which could be expressed using a unit quaternion q), velocity (v_x, v_y, v_z) and angular velocity $(\omega_x, \omega_y, \omega_z)$.

The helicopter is controlled via a 4-dimensional action space:

1. u_1 and u_2: The lateral (left-right) and longitudinal (front-back) cyclic pitch controls (together referred to as the "cyclic" controls) cause the helicopter to roll left or right, and pitch forward or backward, respectively.
2. u_3: The tail rotor pitch control affects tail rotor thrust, and can be used to yaw (turn) the helicopter about its vertical axis. In analogy to airplane control, the tail rotor control is commonly referred to as "rudder."
3. u_4: The collective pitch control (often referred to simply as "collective"), increases and decreases the pitch of the main rotor blades, thus increasing or decreasing the vertical thrust produced as the blades sweep through the air.

By using the cyclic and rudder controls, the pilot can rotate the helicopter into any orientation. This allows the pilot to direct the thrust of the main rotor in any particular direction, and thus fly in any direction, by rotating the helicopter appropriately.

Helicopter Flight as an RL Problem

Formulation

A RL problem can be described by a tuple $(S, \mathcal{A}, T, H, s(0), R)$, which is referred to as a ▶Markov decision process (MDP). Here S is the set of states; $\mathcal{A}$ is the set of actions or inputs; T is the dynamics model, which is a set of probability distributions $\{P^t_{su}\}$ ($P^t_{su}(s'|s,u)$ is the probability of being in state s' at time $t+1$, given the state and action at time t are s and u); H is the horizon or number of time steps of interest; $s(0) \in S$ is the initial state; $R : S \times \mathcal{A} \to \mathbb{R}$ is the reward function.

A policy $\pi = (\mu_0, \mu_1, \ldots, \mu_H)$ is a tuple of mappings from states S to actions $\mathcal{A}$, one mapping for each time $t = 0, \ldots, H$. The expected sum of rewards when acting according to a policy π is given by: $U(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$. The optimal policy π^* for an MDP $(S, \mathcal{A}, T, H, s(0), R)$ is the policy that maximizes the expected sum of rewards. In particular, the optimal policy is given by: $\pi^* = \arg\max_\pi U(\pi)$.

The common approach to finding a good policy for autonomous helicopter flight proceeds in two steps: First one collects data from manual helicopter flights to build a model (One could also build a helicopter model by directly measuring physical parameters such as mass, rotor span, etc. However, even when this approach is pursued, one often resorts to collecting flight data to complete the model.). Then one solves the MDP comprised of the model and some chosen reward function. Although the controller obtained, in principle, is only optimal for the learned simulator model, it has been shown in various settings that optimal controllers perform well even when the model has some inaccuracies (see, e.g., Anderson & Moore, (1989)).

Modeling

One way to create a helicopter model is to use direct knowledge of aerodynamics to derive an explicit mathematical model. This model will depends on a number of parameters that are particular to the helicopter

being flown. Many of the parameters may be measured directly (e.g., mass, rotational inertia), while others must be estimated from flight experiments. This approach has been used successfully on several systems (see, e.g., (Gavrilets, Martinos, Mettler, & Feron, 2002b; Gavrilets, Mettler, & Feron, 2001; La Civita, 2003)). However, substantial expert aerodynamics knowledge is required for this modeling approach. Moreover, these models tend to cover only a limited fraction of the flight envelope.

Alternatively, one can learn a model of the dynamics directly from flight data, with only limited *a priori* knowledge of the helicopter's dynamics. Data is usually collected from a series of manually controlled flights. These flights involve the human sweeping the control sticks back and forth at varying frequencies to cover as much of the flight envelope as possible, while recording the helicopter's state and the pilot inputs at each instant.

Given a corpus of flight data, various different learning algorithms can be used to learn the underlying model of the helicopter dynamics.

If one is only interested in a single flight regime, one could learn a linear model that maps from the current state and action to the next state. Such a model can be easily estimated using ►linear regression (While the methods presented here emphasize time-domain estimation, frequency domain estimation is also possible for the special case of estimating linear models (Tischler & Cauffman, 1992).). Linear models are restricted to small flight regimes (e.g., hover or inverted hover) and do not immediately generalize to full-envelope flight. To cover a broader flight regime, non parametric algorithms such as locally-weighted linear regression have been used (Bagnell & Schneider, 2001; Ng et al., 2004). Non parametric models that map from current state and action to next state can, in principle, cover the entire flight regime. Unfortunately, one must collect large amounts of data to obtain an accurate model and the models are often quite slow to evaluate.

An alternative way to increase the expressiveness of the model, without resorting to non parametric methods, is to consider a time-varying model where the dynamics are explicitly allowed to depend on time. One can then proceed to compute simpler (say, linear) parametric models for each choice of the time parameter. This method is effective when learning a model specific to a trajectory whose dynamics are repeatable but vary as the aircraft travels along the trajectory. Since this method can also require a great deal of data (similar to nonparametric methods) in practice, it is helpful to begin with a non-time-varying parametric model fit from a large amount of data, and then augment it with a time-varying component that has fewer parameters (Abbeel, Quigley, & Ng, 2006; Coates et al., 2008).

One can also take advantage of symmetry in the helicopter dynamics to reduce the amount of data needed to fit a parametric model. In Abbeel, Ganapathi, and Ng (2006) observe that – in a coordinate frame attached to the helicopter – the helicopter dynamics are essentially the same for any orientation (or position) once the effect of gravity is removed. They learn a model that predicts (angular and linear) accelerations – except for the effects of gravity – in the helicopter frame as a function of the inputs and the (angular and linear) velocity in the helicopter frame. This leads to a lower-dimensional learning problem, which requires significantly less data. To simulate the helicopter dynamics over time, the predicted accelerations augmented with the effects of gravity are integrated over time to obtain velocity, angular rates, position, and orientation.

Abbeel et al. (2007) used this approach to learn a helicopter model that was later used for autonomous aerobatic helicopter flight maneuvers covering a large part of the flight envelope. Significantly less data is required to learn a model using the gravity-free parameterization compared to a parameterization that directly predicts the next state as a function of current state and actions (as was used in Bagnell and Schneider (2001), Ng et al. (2004)). Abbeel et al. evaluate their model by checking its simulation accuracy over longer time scales than just a one-step acceleration prediction. Such an evaluation criterion maps more directly to the reinforcement learning objective of maximizing the expected sum of rewards accumulated over time (see also Abbeel & Ng, (2005b)).

The models considered above are deterministic. This normally would allow us to drop the expectation when evaluating a policy according to $E[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$. However, it is common to add stochasticity to account for unmodeled effects. Abbeel et al. (2007) and Ng et al. (2004) include additive process noise in

their models. Bagnell and Schneider (2001) go further, learning a distribution over models. Their policy must then perform well, on expectation, for a (deterministic) model selected randomly from the distribution.

Control Problem Solution Methods

Given a model of the helicopter, we now seek a policy π that maximizes the expected sum of rewards $U(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$ achieved when acting according to the policy π.

Policy Search General policy search algorithms can be employed to search for optimal policies for the MDP based on the learned model. Given a policy π, we can directly try to optimize the objective $U(\pi)$. Unfortunately, $U(\pi)$ is an expectation over a complicated distribution making it impractical to evaluate the expectation exactly in general.

One solution is to approximate the expectation $U(\pi)$ by Monte Carlo sampling: under certain boundedness assumptions the empirical average of the sum of rewards accumulated over time will give a good estimate $\hat{U}(\pi)$ of the expectation $U(\pi)$. Naively Applying Monte Carlo sampling to accurately compute, e.g., the local gradient from the difference in function value at nearby points, requires very large amounts of samples due to the stochasticity in the function evaluation.

To get around this hurdle, the PEGASUS algorithm (Ng & Jordan, 2000) can be used to convert the stochastic optimization problem into a deterministic one. When evaluating by averaging over n simulations, PEGASUS initially fixes n random seeds. For each policy evaluation, the same n random seeds are used so that the simulator is now deterministic. In particular, multiple evaluations of the same policy will result in the same computed reward. A search algorithm can then be applied to the deterministic problem to find an optimum.

The PEGASUS algorithm coupled with a simple local policy search was used by Ng et al. (2004) to develop a policy for their autonomous helicopter that successfully sustains inverted hover. Bagnell and Schneider (2001) proceed similarly, but use the "amoeba" search algorithm (Nelder & Mead, 1964) for policy search.

Because of the searching involved, the policy class must generally have low dimension. Nonetheless, it is often possible to find good policies within these policy classes. The policy class of Ng et al. (2004), for instance, is a decoupled, linear PD controller with a sparse dependence on the state variables (For instance, the linear controller for the pitch axis is parametrized as $u_2 = c_0(p_x - p_x^*) + c_1(v_x - v_x^*) + c_2\theta$, which has just three parameters while the entire state is nine dimensional. Here, $p_\cdot$, $v_\cdot$, and $p_\cdot^*$, $v_\cdot^*$, respectively, are the actual and desired position and velocity. θ denotes the pitch angle.). The sparsity reduces the policy class to just nine parameters. In Bagnell and Schneider (2001), two-layer neural network structures are used with a similar sparse dependence on the state variables. Two neural networks with five parameters each are learned for the cyclic controls.

Differential Dynamic Programming Abbeel et al. (2007) use differential dynamic programming (DDP) for the task of aerobatic trajectory following. DDP (Jacobson & Mayne, 1970) works by iteratively approximating the MDP as linear quadratic regulator (LQR) problems. The LQR control problem is a special class of MDPs, for which the optimal policy can be computed efficiently. In LQR the set of states is given by $S = \mathbb{R}^n$, the set of actions/inputs is given by $\mathcal{A} = \mathbb{R}^p$, and the dynamics model is given by:

$$s(t+1) = A(t)s(t) + B(t)u(t) + w(t),$$

where for all $t = 0, \ldots, H$ we have that $A(t) \in \mathbb{R}^{n \times n}$, $B(t) \in \mathbb{R}^{n \times p}$ and $w(t)$ is a mean zero random variable (with finite variance). The reward for being in state $s(t)$ and taking action $u(t)$ is given by:

$$-s(t)^\top Q(t)s(t) - u(t)^\top R(t)u(t).$$

Here $Q(t), R(t)$ are positive semi-definite matrices which parameterize the reward function. It is well-known that the optimal policy for the LQR control problem is a linear feedback controller which can be efficiently computed using dynamic programming (see, e.g., Anderson & Moore, (1989), for details on linear quadratic methods.)

DDP approximately solves general continuous state-space MDPs by iterating the following two steps until convergence:

1. Compute a linear approximation to the nonlinear dynamics and a quadratic approximation to

the reward function around the trajectory obtained when executing the current policy in simulation.
2. Compute the optimal policy for the LQR problem obtained in Step 1 and set the current policy equal to the optimal policy for the LQR problem.

During the first iteration, the linearizations are performed around the target trajectory for the maneuver, since an initial policy is not available.

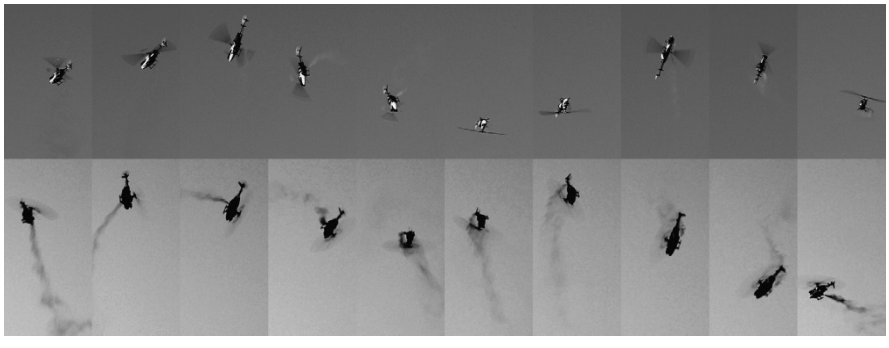
This method is used to perform autonomous flips, rolls, and "funnels" (high-speed sideways flight in a circle) in Abbeel et al. (2007) and autonomous autorotation (Autorotation is an emergency maneuver that allows a skilled pilot to glide a helicopter to a safe landing in the event of an engine failure or tail-rotor failure.) in Abbeel, Coates, Hunter, and Ng (2008), Fig. 2.

While DDP computes a solution to the non-linear optimization problem, it relies on the accuracy of the non-linear model to correctly predict the trajectory that will be flown by the helicopter. This prediction is used in Step 1 above to linearize the dynamics. In practice, the helicopter will often not follow the predicted trajectory closely (due to stochasticity and modeling errors), and thus the linearization will become a highly inaccurate approximation of the non-linear model. A common solution to this, applied by Coates et al. (2008), is to compute the DDP solution online, linearizing around a trajectory that begins at the current helicopter state. This ensures that the model is always linearized around a trajectory near the helicopter's actual flight path.

Apprenticeship Learning and Inverse RL In computing a policy for an MDP, simply finding a solution (using any method) that performs well in simulation may not be enough. One may need to adjust both the model and reward function based on the results of flight testing. Modeling error may result in controllers that fly perfectly in simulation but perform poorly or fail entirely in reality. Because helicopter dynamics are difficult to model exactly, this problem is fairly common. Meanwhile, a poor reward function can result in a controller that is not robust to modeling errors or unpredicted perturbations (e.g., it may use large control inputs that are unsafe in practice). If a human "expert" is available to demonstrate the maneuver, this demonstration flight can be leveraged to obtain a better model and reward function.

The reward function encodes both the trajectory that the helicopter should follow, as well as the trade-offs between different types of errors. If the desired trajectory is infeasible (either in the non-linear simulation or in reality), this results in a significantly more difficult control problem. Also, if the trade-offs are not specified correctly, the helicopter may be unable to compensate for significant deviations from the desired trajectory. For instance, a typical reward function for hovering implicitly specifies a trade-off between position error and orientation error (it is possible to reduce one error, but usually at the cost of increasing the other). If this trade-off is incorrectly chosen, the controller may be pushed off course by wind (if it tries too hard to keep the helicopter level) or, conversely, may tilt the helicopter to an unsafe attitude while trying to correct for a large position error.

We can use demonstrations from an expert pilot to recover both a good choice for the desired trajectory as well as good choices of reward weights for errors relative to this trajectory. In apprenticeship learning, we are given a set of N recorded state and control sequences,



Autonomous Helicopter Flight Using Reinforcement Learning. Figure 2. Snapshots of an autonomous helicopter performing in-place flips and rolls

$\{s_k(t), u_k(t)\}_{t=0}^{H}$ for $k = 1, \ldots, N$, from demonstration flights by an expert pilot. Coates et al. (2008) note that these demonstrations may be sub-optimal but are often sub-optimal in different ways. They suggest that a large number of expert demonstrations may implicitly encode the optimal trajectory and propose a generative model that explains the expert demonstrations as stochastic instantiations of an "ideal" trajectory. This is the desired trajectory that the expert has in mind but is unable to demonstrate exactly. Using an Expectation-Maximization (Dempster, Laird, & Rubin, 1977) algorithm, they infer the desired trajectory and use this as the target trajectory in their reward function.

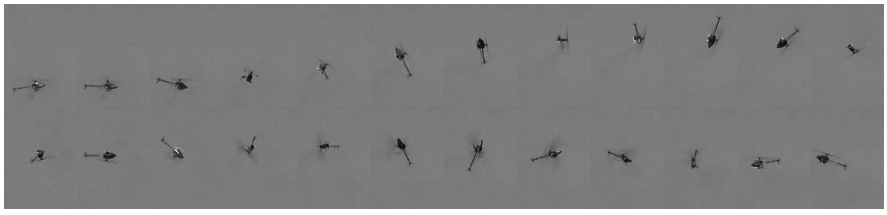
A good choice of reward weights (for errors relative to the desired trajectory) can be recovered using inverse reinforcement learning (Abbeel & Ng, 2004; Ng & Russell, 2000). Suppose the reward function is written as a linear combination of features as follows: $R(s,u) = c_0\phi_0(s,u) + c_1\phi_1(s,u) + \cdots$. For a single recorded demonstration, $\{s(t), u(t)\}_{t=0}^{H}$, the pilot's accumulated reward corresponding to each feature can be computed as $c_i\phi_i^* = c_i \sum_{t=0}^{H} \phi_i(s(t), u(t))$. If the pilot out-performs the autonomous flight controller with respect to a particular feature ϕ_i, this indicates that the pilot's own "reward function" places a higher value on that feature, and hence its weight c_i should be increased. Using this procedure, a good choice of reward function that makes trade-offs similar to that of a human pilot can be recovered. This method has been used to guide the choice of reward for many maneuvers during flight testing (Abbeel et al., 2007, 2008; Coates et al., 2008).

In addition to learning a better reward function from pilot demonstration, one can also use the pilot demonstration to improve the model directly and attempt to reduce modeling error. Coates et al. (2008), for instance, use errors observed in expert demonstrations to jointly infer an improved dynamics model along with the desired trajectory. Abbeel et al. (2007), however, have proposed the following alternating procedure that is broadly applicable (see also Abbeel and Ng (2005a) for details):

1. Collect data from a human pilot flying the desired maneuvers with the helicopter. Learn a model from the data.
2. Find a controller that works in simulation based on the current model.
3. Test the controller on the helicopter. If it works, we are done. Otherwise, use the data from the test flight to learn a new (improved) model and go back to Step 2.

This procedure has similarities with model-based RL and with the common approach in control to first perform system identification and then find a controller using the resulting model. However, the key insight from Abbeel and Ng (2005a) is that this procedure is guaranteed to converge to expert performance in a polynomial number of iterations. The authors report needing at most three iterations in practice. Importantly, unlike the E^3 family of algorithms (Kearns & Singh, 2002), this procedure does not require explicit exploration policies. One only needs to test controllers that try to fly as well as possible (according to the current choice of dynamics model) (Indeed, the E^3-family of algorithms (Kearns & Singh, 2002) and its extensions (Brafman & Tennenholtz, 2002; Kakade, Kearns, & Langford, 2003; Kearns & Koller, 1999) proceed by generating "exploration" policies, which try to visit inaccurately modeled parts of the state space. Unfortunately, such exploration policies do not even try to fly the helicopter well, and thus would almost invariably lead to crashes.).

The apprenticeship learning algorithms described above have been used to fly the most advanced autonomous maneuvers to date. The apprenticeship learning algorithm of Coates et al. (2008), for example, has been used to attain expert level performance on challenging aerobatic maneuvers as well as entire airshows composed of many maneuvers in rapid sequence. These maneuvers include in-place flips and rolls, tic-tocs ("Tic-toc" is a maneuver where the helicopter pitches forward and backward with its nose pointed toward the sky (resembling an inverted clock pendulum).), and chaos ("Chaos" is a maneuver where the helicopter flips in-place but does so while continuously pirouetting at a high rate. Visually, the helicopter body appears to tumble chaotically while nevertheless remaining in roughly the same position.) (see Fig. 3). These maneuvers are considered among the most challenging possible and can only be performed



Autonomous Helicopter Flight Using Reinforcement Learning. Figure 3. Snapshot sequence of an autonomous helicopter flying a "chaos" maneuver using apprenticeship learning methods. Beginning from top-left and proceeding left-to-right, top-to-bottom, the helicopter performs a flip while pirouetting counter-clockwise about its vertical axis. (This maneuver has been demonstrated continuously for as long as 15 cycles like the one shown here)

Autonomous Helicopter Flight Using Reinforcement Learning. Figure 4. Super-imposed sequence of images of autonomous autorotation landings (from Abbeel et al. (2008))

by advanced human pilots. In fact, Coates et al. (2008) show that their learned controller performance can even exceed the performance of the expert pilot providing the demonstrations, putting many of the maneuvers on par with professional pilots (Fig. 4).

A similar approach has been used in Abbeel et al. (2008) to perform the first successful autonomous autorotations. Their aircraft has performed more than 30 autonomous landings successfully without engine power.

Not only do apprenticeship methods achieve state-of-the-art performance, but they are among the fastest learning methods available, as they obviate the need for arduous hand tuning by engineers. Coates et al. (2008), for instance, report that entire airshows can be created from scratch with just 1 h of work. This is in stark contrast to previous approaches that may have required hours or even days of tuning for relatively simple maneuvers.

Conclusion

Helicopter control is a challenging control problem and has recently seen major successes with the application of learning algorithms. This Chapter has shown how each step of the control design process can be automated using machine learning algorithms for system identification and reinforcment learning algorithms for control. It has also shown how apprenticeship learning algorithms can be employed to achieve

expert-level performance on challenging aerobatic maneuvers when an expert pilot can provide demonstrations. Autonomous helicopters with control systems developed using these methods are now capable of flying advanced aerobatic maneuvers (including flips, rolls, tic-tocs, chaos, and auto-rotation) at the level of expert human pilots.

Cross References

►Apprenticeship Learning
►Reinforcement Learning
►Reward Shaping

Recommended Reading

Abbeel, P., Coates, A., Hunter, T., & Ng, A. Y. (2008). Autonomous autorotation of an rc helicopter. In *ISER 11*.

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *NIPS 19* (pp. 1–8). Vancouver.

Abbeel, P., Ganapathi, V., & Ng, A. Y. (2006). Learning vehicular dynamics with application to modeling helicopters. In *NIPS 18*. Vancouver.

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the international conference on machine learning*. New York: ACM.

Abbeel, P., & Ng, A. Y. (2005a). Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the international conference on machine learning*. New York: ACM

Abbeel, P., & Ng, A. Y. (2005b). Learning first order Markov models for control. In *NIPS 18*.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on machine learning* (pp. 1–8). New York: ACM.

Anderson, B., & Moore, J. (1989). *Optimal control: linear quadratic methods*. Princeton, NJ: Prentice-Hall.

Bagnell, J., & Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *International conference on robotics and automation*. Canada: IEEE.

Brafman, R. I., & Tennenholtz, M. (2002). R-max, a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3*, 213–231.

Coates, A., Abbeel, P., & Ng, A. Y. (2008). Learning for control from multiple demonstrations. In *ICML '08: Proceedings of the 25th international conference on machine learning*.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*1, 1–38.

Dunbabin, M., Brosnan, S., Roberts, J., & Corke, P. (2004). Vibration isolation for autonomous helicopter flight. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 4, pp. 3609–3615).

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002a). Control logic for automated aerobatic flight of miniature helicopter. In *AIAA guidance, navigation and control conference*. Cambridge, MA: Massachusetts Institute of Technology.

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002b). Flight test and simulation results for an autonomous aerobatic helicopter. In *AIAA/IEEE digital avionics systems conference*.

Gavrilets, V., Mettler, B., & Feron, E. (2001). Nonlinear model for a small-size acrobatic helicopter. In *AIAA guidance, navigation and control conference* (pp. 1593–1600).

Jacobson, D. H., & Mayne, D. Q. (1970). *Differential dynamic programming*. New York: Elsevier.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. In *Proceedings of the international conference on machine learning*.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th international joint conference on artificial intelligence*. San Francisco: Morgan Kaufmann.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning Journal, 49*(2–3), 209–232.

La Civita, M. (2003). *Integrated modeling and robust control for full-envelope flight of robotic helicopters*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

La Civita, M., Papageorgiou, G., Messner, W. C., & Kanade, T. (2006). Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. *Journal of Guidance, Control, and Dynamics, 29*(2), 485–494.

Leishman, J. (2000). *Principles of helicopter aerodynamics*. Cambridge: Cambridge University Press.

Nelder, J. A., & Mead, R. (1964). A simplex method for function minimization. *The Computer Journal, 7*, 308–313.

Ng, A. Y., & Jordan, M. (2000). Pegasus: A policy search method for large MDPs and POMDPs. In *Proceedings of the uncertainty in artificial intelligence 16th conference*. San Francisco: Morgan Kaufmann.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Procedings of the 17th international conference on machine learning* (pp. 663–670). San Francisco: Morgan Kaufmann.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., et al., (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International symposium on experimental robotics*. Berlin: Springer.

Ng, A. Y., Kim, H. J., Jordan, M., & Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In *NIPS 16*.

Saripalli, S., Montgomery, J. F., & Sukhatme, G. S. (2003). Visually-guided landing of an unmanned aerial vehicle. *IEEE Transactions on Robotics and Autonomous Systems, 19*(3), 371–380.

Seddon, J. (1990). *Basic helicopter aerodynamics*. In AIAA education series. El Segundo, CA: America Institute of Aeronautics and Astronautics.

Tischler, M. B., & Cauffman, M. G. (1992). Frequency response method for rotorcraft system identification: Flight application to BO-105 couple rotor/fuselage dynamics. *Journal of the American Helicopter Society, 37*.

Average-Cost Neuro-Dynamic Programming

►Average-Reward Reinforcement Learning

Average-Cost Optimization

►Average-Reward Reinforcement Learning

Averaged One-Dependence Estimators

Fei Zheng, Geoffrey I. Webb
Monash University

Synonyms

AODE

Definition

Averaged one-dependence estimators is a ►semi-naive Bayesian Learning method. It performs classification by aggregating the predictions of multiple one-dependence classifiers in which all attributes depend on the same single parent attribute as well as the class.

Classification with AODE

An effective approach to accommodating violations of naive Bayes' attribute independence assumption is to allow an attribute to depend on other non-class attributes. To maintain efficiency it can be desirable to utilize one-dependence classifiers, such as ►Tree Augmented Naive Bayes (TAN), in which each attribute depends upon the class and at most one other attribute. However, most approaches to learning with one-dependence classifiers perform model selection, a process that usually imposes substantial computational overheads and substantially increases variance relative to naive Bayes.

AODE avoids model selection by averaging the predictions of multiple one-dependence classifiers. In each one-dependence classifier, an attribute is selected as the parent of all the other attributes. This attribute is called the *SuperParent* and this type of one-dependence classifier is called a *SuperParent one-dependence estimator* (SPODE). Only those SPODEs with SuperParent x_i where the value of x_i occurs at least m times are used for predicting a class label y for the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$. For any attribute value x_i,

$$P(y, \mathbf{x}) = P(y, x_i) P(\mathbf{x} \mid y, x_i).$$

This equality holds for every x_i. Therefore,

$$P(y, \mathbf{x}) = \frac{\sum_{1 \le i \le n \wedge F(x_i) \ge m} P(y, x_i) P(\mathbf{x} \mid y, x_i)}{|\{1 \le i \le n \wedge F(x_i) \ge m\}|}, \quad (1)$$

where $F(x_i)$ is the frequency of attribute value x_i in the training sample. Utilizing (1) and the assumption that attributes are independent given the class and the SuperParent x_i, AODE predicts the class for $\mathbf{x}$ by selecting

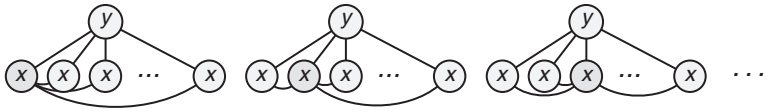
$$\operatorname*{argmax}_y \sum_{1 \le i \le n \wedge F(x_i) \ge m} \hat{P}(y, x_i) \prod_{1 \le j \le n, j \ne i} \hat{P}(x_j \mid y, x_i). \quad (2)$$

It averages over estimates of the terms in (1), rather than the true values, which has the effect of reducing the variance of these estimates.

Figure 1 shows a Markov network representation of an example AODE.

As AODE makes a weaker attribute conditional independence assumption than naive Bayes while still avoiding model selection, it has substantially lower ►bias with a very small increase in ►variance. A number of studies (Webb, Boughton, & Wang, 2005; Zheng & Webb, 2005) have demonstrated that it often has considerably lower zero-one loss than naive Bayes with moderate time complexity. For comparisons with other semi-naive techniques, see ►semi-naive Bayesian learning. One study (Webb, Boughton, & Wang, 2005) found AODE to provide classification accuracy competitive to a state-of-the-art discriminative algorithm, boosted decision trees.

When a new instance is available, like naive Bayes, AODE only needs to update the probability estimates. Therefore, it is also suited to incremental learning.



Averaged One-Dependence Estimators. Figure 1. A Markov network representation of the SPODEs that comprise an example AODE

Cross References

►Bayesian Network
►Naive Bayes
►Semi-Naive Bayesian Learning
►Tree-Augmented Naive Bayes

Recommended Reading

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: aggregating one-dependence estimators. *Machine Learning, 58*(1), 5–24.

Zheng, F., & Webb, G. I. (2005). A comparative study of semi-naive Bayes methods in classification learning. In *Proceedings of the Fourth Australasian Data Mining Conference*. (pp. 141–156).

Average-Payoff Reinforcement Learning

►Average-Reward Reinforcement Learning

Average-Reward Reinforcement Learning

Prasad Tadepalli
Oregon State University, Corvallis, OR, USA

Synonyms

ARL; Average-cost neuro-dynamic programming; Average-cost optimization; Average-payoff reinforcement learning

Definition

Average-reward reinforcement learning (ARL) refers to learning policies that optimize the average reward per time step by continually taking actions and observing the outcomes including the next state and the immediate reward.

Motivation and Background

►Reinforcement learning (RL) is the study of programs that improve their performance at some task by receiving rewards and punishments from the environment (Sutton & Barto, 1998). RL has been quite successful in automatic learning of good procedures for complex tasks such as playing Backgammon and scheduling elevators (Crites & Barto, 1998; Tesauro, 1992). In episodic domains in which there is a natural termination condition such as the end of the game in Backgammon, the obvious performance measure to optimize is the expected total reward per episode. But some domains such as elevator scheduling are recurrent, i.e., do not have a natural termination condition. In such cases, total expected reward can be infinite, and we need a different optimization criterion.

In the discounted optimization framework, in each time step, the value of the reward is multiplied by a discount factor $\gamma < 1$, so that the total *discounted* reward is always finite. However, in many domains, there is no natural interpretation for the discount factor γ. A natural performance measure to optimize in such domains is the average reward received per time step. Although one could use a discount factor which is close to 1 to approximate average-reward optimization, an approach that directly optimizes the average reward avoids this additional parameter and often leads to faster convergence in practice.

There is significant theory behind average-reward optimization based on ►Markov decision processes (MDPs) (Puterman, 1994). An MDP is described by a 4-tuple $\langle S, A, P, r \rangle$, where S is a discrete set of states and A is a discrete set of actions. P is a conditional probability distribution over the next states, given the current state and action, and r gives the immediate reward for a given state and action. A *policy* π is a mapping from states to actions. Each policy π induces a Markov process over some set of states. In ergodic MDPs, every policy π forms a single closed set of states, and the average reward per time step of π in the limit of infinite

horizon is independent of the starting state. We call it the "gain" of the policy π, denoted by $\rho(\pi)$, and consider the problem of finding a "gain-optimal policy," π^*, that maximizes $\rho(\pi)$.

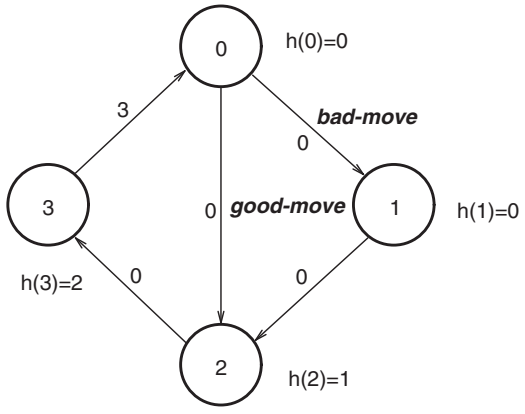
Even though the gain $\rho(\pi)$ of a policy π is independent of the starting state s, the total expected reward in time t is not. It can be denoted by $\rho(\pi)t + h(s)$, where $h(s)$ is a state-dependent bias term. It is the bias values of states that determine which states and actions are preferred, and need to be learned for optimal performance. The following theorem gives the Bellman equation for the bias values of states.

Theorem 3 *For ergodic MDPs, there exist a scalar ρ and a real-valued bias function h over S that satisfy the recurrence relation*

$$\forall s \in S, \quad h(s) = \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a) h(s') \right\} - \rho. \tag{1}$$

Further, the gain-optimal policy μ^* attains the above maximum for each state s, and ρ is its gain.

Note that any one solution to (1) yields an infinite number of solutions by adding the same constant to all h-values. However, all these sets of h-values will result in the same set of optimal policies μ^*, since the optimal action in a state is determined only by the relative differences between the values of h.



Average-Reward Reinforcement Learning. Figure 1. A simple Markov decision process (MDP) that illustrates the Bellman equation

For example, in Fig. 1, the agent has to select between the actions good-move and bad-move in state 0. If it stays in state 1, it gets an average reward of 1. If it stays in state 2, it gets an average reward of −1. For this domain, $\rho = 1$ for the optimal policy of choosing good-move in state 0. If we arbitrarily set $h(0)$ to 0, then $h(1) = 0$, $h(2) = 1$, and $h(3) = 2$ satisfy the recurrence relations in (1). For example, the difference between $h(3)$ and $h(1)$ is 2, which equals the difference between the immediate reward for the optimal action in state 3 and the optimal average reward 1.

Given the probability model P and the immediate rewards r, the above equations can be solved by White's relative value iteration method by setting the h-value of an arbitrarily chosen reference state to 0 and using synchronous successive approximation (Bertsekas, 1995). There is also a policy iteration approach to determine the optimal policy starting with some arbitrary policy, solving for its values using the value iteration, and updating the policy using one step look-ahead search. The above iteration is repeated until the policy converges (Puterman, 1994).

Model-Based Learning

If the probabilities and the immediate rewards are not known, the system needs to learn them before applying the above methods. A model-based approach called H-learning interleaves model learning with Bellman backups of the value function (Tadepalli & Ok, 1998). This is an average-reward version of ▶adaptive real-time dynamic programming (Barto, Bradtke, & Singh, 1995). The models are learned by collecting samples of state-action-next-state triples $\langle s, a, s' \rangle$ and computing $P(s'|s,a)$ using the maximum likelihood estimation. It then employs the "certainty equivalence principle" by using the current estimates as the true value while updating the h-value of the current state s according to the following update equation derived from the Bellman equation.

$$h(s) \leftarrow \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a) h(s') \right\} - \rho. \tag{2}$$

One complication in ARL is the estimation of the average reward ρ in the update equations during learning. One could use the current estimate of the long-term average reward, but it is distorted

by the exploratory actions that the agent needs to take to learn about the unexplored parts of the state space. Without the exploratory actions, ARL methods converge to a suboptimal policy. To take this into account, we have from (1), in any state s and a non-exploratory action a that maximizes the right-hand side, $\rho = r(s,a) - h(s) + \sum_{s' \in S} P(s'|S,a)h(s')$. Hence, ρ is estimated by cumulatively averaging $r - h(s) + h(s')$, whenever a greedy action a is executed in state s resulting in state s' and immediate reward r. ρ is updated using the following equation where α is the learning rate.

$$\rho \leftarrow \rho + \alpha(r - h(s) + h(s')). \quad (3)$$

One issue with model-based learning is that the models require too much space and time to learn as tables. In many cases, actions can be represented much more compactly. For example, Tadepalli and Ok (1998) uses dynamic Bayesian networks to represent and learn action models, resulting in significant savings in space and time for learning the models.

Model-Free Learning

One of the disadvantages of the model-based methods is the need to explicitly represent and learn action models. This is completely avoided in model-free methods such as ►Q-learning by learning value functions over state–action pairs. Schwartz's R-learning is an adaptation of Q-learning, which is a discounted reinforcement learning method, to optimize average reward (Schwartz, 1993).

The state–action value $R(s,a)$ can be defined as the expected long-term advantage of executing action a in state s and from then on following the optimal average-reward policy. It can be defined using the bias values h and the optimal average reward ρ as follows.

$$R(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s') - \rho. \quad (4)$$

The main difference with Q-values is that instead of discounting the expected total reward from the next state, we subtract the average reward ρ in each time step, which is the constant penalty for using up a time step. The h value of any state s can now be defined using the following equation.

$$h(s') = \max_u R(s',u). \quad (5)$$

Initially all the R-values are set to 0. When action a is executed in state s, the value of $R(s,a)$ is updated using the update equation

$$R(s,a) \leftarrow (1-\beta)R(s,a) + \beta(r + h(s') - \rho), \quad (6)$$

where β is the learning rate, r is the immediate reward received, s' is the next state, and ρ is the estimate of the average reward of the current greedy policy. In any state s, the greedy action a maximizes the value $R(s,a)$; so R-learning does not need to explicitly learn the immediate reward function $r(s,a)$ or the action models $P(s'|s,a)$, since it does not use them either for the action selection or for updating the R-values.

Both model-free and model-based ARL methods have been evaluated in several experimental domains (Mahadevan, 1996; Tadepalli & Ok, 1998). When there is a compact representation for models and can be learned quickly, the model-based method seems to perform better. It also has the advantage of fewer number of tunable parameters. However, model-free methods are more convenient to implement especially if the models are hard to learn or represent.

Scaling Average-Reward Reinforcement Learning

Just as for discounted reinforcement learning, scaling issues are paramount for ARL. Since the number of states is exponential to the number of relevant state variables, a table-based approach does not scale well. The problem is compounded in multi-agent domains where the number of joint actions is exponential in the number of agents. Several function approximation approaches, such as linear functions, multi-layer perceptrons (Marbach, Mihatsch, & Tsitsiklis, 2000), local ►linear regression (Tadepalli & Ok, 1998), and tile coding (Proper & Tadepalli, 2006) were tried with varying degrees of success.

►Hierarchical reinforcement learning based on the MAXQ framework was also explored in the average-reward setting and was shown to lead to significantly faster convergence. In MAXQ framework, we have a directed acyclic graph, where each node represents a task and stores the value function for that task. Usually, the value function for subtasks depends on fewer state variables than the overall value function and hence can

be more compactly represented. The relevant variables for each subtask are fixed by the designer of the hierarchy, which makes it much easier to learn the value functions. One potential problem with the hierarchical approach is the loss due to the hierarchical constraint on the policy. Despite this limitation, both model-based (Seri & Tadepalli, 2002) and model-free approaches (Ghavamzadeh & Mahadevan, 2006) were shown to yield optimal policies in some domains that satisfy the assumptions of these methods.

Applications

A temporal difference method for average reward based on TD(0) was used to solve a call admission control and routing problem (Marbach et al., 2000). On a modestly sized network of 16 nodes, it was shown that the average-reward TD(0) outperforms the discounted version because it required more careful tuning of its parameters. Similar results were obtained in other domains such as automatic guided vehicle routing (Ghavamzadeh & Mahadevan, 2006) and transfer line optimization (Wang & Mahadevan, 1999).

Convergence Analysis

Unlike their discounted counterparts, both R-Learning and H-Learning lack convergence guarantees. This is because due to the lack of discounting, the updates can no longer be thought of as contraction mappings, and hence the standard theory of stochastic approximation does not apply. Simultaneous update of the average reward ρ and the value functions makes the analysis of these algorithms much more complicated. However, some ARL algorithms have been proved convergent in the limit using analysis based on ordinary differential equations (ODE) (Abounadi, Bertsekas, & Borkar, 2002). The main idea is to turn to ordinary differential equations that are closely tracked by the update equations and use two time-scale analysis to show convergence. In addition to the standard assumptions of stochastic approximation theory, the two time-scale analysis requires that ρ is updated at a much slower time scale than the value function.

The previous convergence results are based on the limit of infinite exploration. One of the many challenges in reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. There are model-based algorithms that guarantee learning an approximately optimal average-reward policy in time polynomial in the numbers of states and actions of the MDP and its mixing time. These algorithms work by alternating between learning the action models of the MDP by taking actions in the environment, and solving the learned MDP using offline value iteration.

In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns & Singh, 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-Max algorithm implicitly chooses between exploration and exploitation by using the principle of "*optimism under uncertainty*" (Brafman & Tennenholtz, 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. The optimistic initialization of the model parameters automatically encourages the agent to execute unexplored actions, until the true models and values of more states and actions are gradually revealed to the agent. It has been shown that with a probability at least $1 - \delta$, both E^3 and R-MAX learn approximately correct models whose optimal policies have an average reward ϵ-close to the true optimal in time polynomial in the numbers of states and actions, the mixing time of the MDP, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$.

Unfortunately the convergence results do not apply when there is function approximation involved. In the presence of linear function approximation, the average-reward version of temporal difference learning, which learns a state-based value function for a fixed policy, is shown to converge in the limit (Tsitsiklis & Van Roy, 1999). The transient behavior of this algorithm is similar to that of the corresponding discounted TD-learning with an appropriately scaled constant basis function (Van Roy & Tsitsiklis, 2002). As in the discounted case, development of provably convergent optimal policy learning algorithms with function approximation is a challenging open problem.

Cross References

►Efficient Exploration in Reinforcement Learning
►Hierarchical Reinforcement Learning
►Model-Based Reinforcement Learning

Recommended Reading

Abounadi, J., Bertsekas, D. P., & Borkar, V. (2002). Stochastic approximation for non-expansive maps: Application to Q-learning algorithms. *SIAM Journal of Control and Optimization, 41*(1), 1–22.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1), 81–138.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 2*, 213–231.

Crites, R. H., & Barto, A. G. (1998). Elevator group control using multiple reinforcement agents. *Machine Learning, 33*(2/3), 235–262.

Ghavamzadeh, M., & Mahadevan, S. (2006). Hierarchical average reward reinforcement learning. *Journal of Machine Learning Research, 13*(2), 197–229.

Kearns, M., & Singh S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2/3), 209–232.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning, 22*(1/2/3), 159–195.

Marbach, P., Mihatsch, O., & Tsitsiklis, J. N. (2000). Call admission control and routing in integrated service networks using neuro-dynamic programming. *IEEE Journal on Selected Areas in Communications, 18*(2), 197–208.

Proper, S., & Tadepalli, P. (2006). Scaling model-based average-reward reinforcement learning for product delivery. In *European conference on machine learning* (pp. 725–742). Springer.

Puterman, M. L. (1994). *Markov decision processes: Discrete dynamic stochastic programming*. New York: Wiley.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the tenth international conference on machine learning* (pp. 298–305). San Mateo, CA: Morgan Kaufmann.

Seri, S., & Tadepalli, P. (2002). Model-based hierarchical average-reward reinforcement learning. In *Proceedings of international machine learning conference* (pp. 562–569). Sydney, Australia: Morgan Kaufmann.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tadepalli, P., & Ok, D. (1998). Model-based average-reward reinforcement learning. *Artificial Intelligence, 100*, 177–224.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*(3–4), 257–277.

Tsitsiklis, J., & Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica, 35*(11), 1799–1808.

Van Roy, B., & Tsitsiklis, J. (2002). On average versus discounted temporal-difference learning. *Machine Learning, 49*(2/3), 179–191.

Wang, G., & Mahadevan, S. (1999). Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proceedings of the 16th international conference on machine learning* (pp. 464–473). Bled, Slovenia.