A

A/B Testing

▶ Online Controlled Experiments and A/B Testing

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 et al. (1994). Another early link of abduction to learning was given by the ▶ explanation based learning method (DeJong and Mooney 1986), where the abductive explanations of the learning data (training examples) are generalized to all cases. An extensive survey of the role of abduction in Machine Learning during this early period can be found in Bergadano et al. (2000).

Following this, it was realized (Flach and 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

© Springer Science+Business Media New York 2017
C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

(Muggleton and Bryant 2000) and HAIL (Ray et al. 2003). On the other hand, from the point of view of abduction as "inference to the best explanation" (Josephson and 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 et al. 2005; Ray et al. 2006; Tamaddoni-Nezhad et al. 2004; Zupan et al. 2003).

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 set covering (See, e.g., Reggia 1983) or case-based explanation, (e.g., Leake 1995). The following explanation uses a logic-based approach.

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 et al. 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, $\mathcal{T} = \langle T, A, \mathrm{IC}\rangle$, and an observation O then, O is *abductively entailed* by $\mathcal{T}$, denoted by $\mathcal{T} \models_A O$, if there exists an abductive explanation of O in $\mathcal{T}$.

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 and 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, $\langle T, A, \mathrm{IC}\rangle$ 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 1 Suppose we want to learn the concept $father$. Let the background theory be $T = \langle P, A, \emptyset \rangle$ 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.

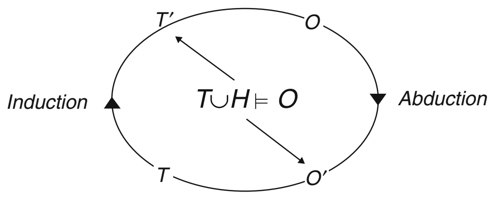
Abduction and Induction

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 and 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



Abduction, Fig. 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 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 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:

$$sad(X) \leftarrow tired(X), poor(X),$$

$$tired(oli), tired(ale), tired(kr),$$

$$academic(oli), academic(ale), academic(kr),$$

$$student(oli), lecturer(ale), lecturer(kr),$$

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:

$$poor(X) \leftarrow lecturer(X)$$

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 et al. 2006, 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 and 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 et al. 2009) and complex scientific theories for systems biology (Ray and 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 and Kakas 2009) and 2009 (Ray et al. 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 et al. (2007), respectively. Finally, the recent book edited by Cerro and Inoue (2014) on the logical modeling of biological systems contains several articles on the application of abduction in systems biology.

Cross-References

▶ Explanation-Based Learning
▶ Inductive Logic Programming

Recommended Reading

Ade H, Denecker M (1995) AILP: abductive inductive logic programming. In: Mellish CS (ed) IJCAI. Morgan Kaufmann, San Francisco, pp 1201–1209

Ade H, Malfait B, Raedt LD (1994) Ruth: an ILP theory revision system. In: ISMIS94. Springer, Berlin

Alrajeh D, Ray O, Russo A, Uchitel S (2009) Using abduction and induction for operational requirements elaboration. J Appl Logic 7(3):275–288

Bergadano F, Cutello V, Gunetti D (2000) Abduction in machine learning. In: Gabbay D, Kruse R (eds) Handbook of defeasible reasoning and uncertainty management systems, vol 4. Kluver Academic Press, Dordrecht, pp 197–229

del Cerro LF, Inoue K (eds) (2014) Logical modeling of biological systems. Wiley/ISTE, Hoboken/London

DeJong G, Mooney R (1986) Explanation-based learning: an alternate view. Mach Learn 1:145–176

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

Flach P, Kakas A (2000) Abductive and inductive reasoning: background and issues. In: Flach PA, Kakas AC (eds) Abductive and inductive reasoning. Pure and applied logic. Kluwer, Dordrecht

Flach PA, Kakas AC (eds) (2009) Abduction and induction in artificial intelligence [special issue]. J Appl Logic 7(3):251

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. Springer, Berlin, pp 303–314

Josephson J, Josephson S (eds) (1994) Abductive inference: computation, philosophy, technology. Cambridge University Press, New York

Kakas A, Kowalski R, Toni F (1992) Abductive logic programming. J Logic Comput 2(6):719–770

Kakas A, Riguzzi F (2000) Abductive concept learning. New Gener Comput 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. J Exp Theor Artif Intell 7:407–428

Michalski RS (1993) Inferential theory of learning as a conceptual basis for multistrategy learning. Mach Learn 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. Springer, Berlin, pp 182–197

Moyle SA (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 Gener Comput 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). Springer, Berlin, pp 130–146

Ourston D, Mooney RJ (1994) Theory refinement combining analytical and empirical methods. Artif Intell 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. Springer, Berlin, pp389–393

Ray O (2009) Nonmonotonic abductive inductive learning. J Appl 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: Brewka G, Coradeschi S, Perini A, Traverso P (eds) Proceedings of the 17th European conference on artificial intelligence. Frontiers in artificial intelligence and applications, vol 141. IOS Press, Amsterdam, pp 437–441

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. Springer, Berlin, pp 311–328

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. IEEE Computer Society, Washington, DC, pp 667–671

Ray O, Flach PA, Kakas AC (eds) (2009) Abduction and induction in artificial intelligence. In: Proceedings of IJCAI 2009 workshop

Reggia J (1983) Diagnostic experts systems based on a set-covering model. Int J Man-Mach Stud 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. Mach Learn 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. Springer, Berlin, pp 305–322

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. Springer, Berlin, pp 296–308

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 matches 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
▶ Mean Absolute Error
▶ Model Evaluation
▶ Resubstitution Estimate

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
Edinburgh, UK

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 and 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 and 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 and 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 et al. 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 and 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 and Koller 2001).
4. *Loss minimization* (Cohn 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 1996). The process is usually as follows:

1. Begin by requesting labels for a small random subsample of the examples x_1, x_2, 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 expec-

tation $\hat{y}(x)$ 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 $p(\tilde{y}|\tilde{y})$. 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 $p(\tilde{y}|\tilde{y})$ and its effect on $\sigma^2_{\hat{y}(x)}$ in closed form (Cohn 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 M candidates, 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 et al. 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:

$$\mu_x = \frac{\Sigma_i k_i x_i}{n}, \quad \sigma^2_x = \frac{\Sigma_i k_i (x_i - \mu_x)^2}{n},$$

$$\sigma_{xy} = \frac{\Sigma_i k_i (x_i - \mu_x)(y_i - \mu_y)}{n}$$

$$\mu_y = \frac{\Sigma_i k_i y_i}{n}, \quad \sigma^2_y = \frac{\Sigma_i k_i (y_i - \mu_y)^2}{n},$$

$$\sigma^2_{y|x} = \sigma^2_y - \frac{\sigma_{xy}}{\sigma^2_x}$$

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

$$\hat{y} = \mu_y + \frac{\sigma_{xy}}{\sigma^2_x}(x - \mu_x), \sigma^2_{\hat{y}} = \frac{\sigma^2_{y|x}}{n^2}$$

$$\times \left(\sum_i k_i^2 + \frac{(x-\mu_x)^2}{\sigma^2_x} \sum_i k_i^2 \frac{(x_i - \mu_x)^2}{\sigma^2_x} \right).$$

Our proxy for model error is the variance of our prediction, integrated over the test distribution $\left\langle \sigma^2_{\hat{y}} \right\rangle$. 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 $\left\langle \tilde{\sigma}^2_{\hat{y}} \right\rangle$. While we do not *know* what value $\tilde{y}$ will take, our model gives us an estimated mean $\hat{y}(\tilde{x})$ and variance $\sigma^2_{\hat{y}(\tilde{x})}$ 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 $\left\langle \tilde{\sigma}^2_{\hat{y}} \right\rangle$ in closed form. Defining $\tilde{k}$ as $K(\tilde{x}, x)$, we write:

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

where the component expectations are computed as follows:

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

$$\left\langle \tilde{\sigma}^2_y \right\rangle = \frac{n\sigma^2_y}{n+\tilde{k}} + \frac{n\tilde{k}(\sigma^2_{y1\overline{x}} + (\hat{y}(\tilde{x})-\mu_y)^2)}{(n+\tilde{k})^2},$$

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

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

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

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

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. Inf Comput 75(2):87–106

Angluin D (1988) Queries and concept learning. Mach Learn 2:319–342

Box GEP, Draper N (1987) Empirical model-building and response surfaces. Wiley, New York

Cleveland W, Devlin S, Gross E (1988) Regression by local fitting. J Econom 37:87–114

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

Cohn D, Ghahramani Z, Jordan MI (1996) Active learning with statistical models. J Artif Intell Res 4:129–145. http://citeseer.ist.psu.edu/321503.html

Dasgupta S (1999) Learning mixtures of Gaussians. Found Comput Sci 634–644
Fedorov V (1972) Theory of optimal experiments. Academic Press, New York
Kearns M, Li M, Pitt L, Valiant L (1987) On the learnability of Boolean formulae. In: Proceedings of the 19th annual ACM conference on theory of computing. ACM Press, New York, pp 285–295
Lewis DD, Gail WA (1994) A sequential algorithm for training text classifiers. In: Proceedings of the 17th annual international ACM SIGIR conference, Dublin, pp 3–12
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), Madison, pp 359–367
North DW (1968) A tutorial introduction to decision theory. IEEE Trans Syst Sci Cybern 4(3)
Pitt L, Valiant LG (1988) Computational limitations on learning from examples. J ACM (JACM) 35(4):965–984
Robbins H (1952) Some aspects of the sequential design of experiments. Bull Am Math Soc 55:527–535
Ruff R, Dietterich T (1989) What good are experiments? In: Proceedings of the sixth international workshop on machine learning, Ithaca
Seung HS, Opper M, Sompolinsky H (1992) Query by committee. In: Proceedings of the fifth workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 287–294
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 and 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.

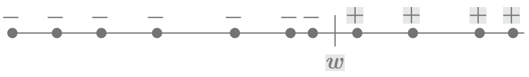
Motivating Examples

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 $\text{err}_{\mathbb{P}} = \mathbb{P}(h(X) \neq Y)$, is at most some ϵ. How many labels do we need?

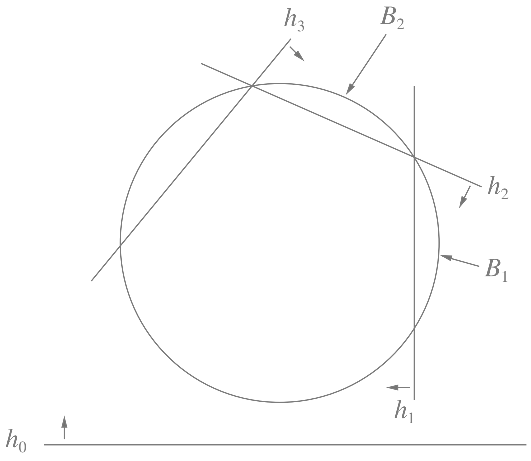
In supervised learning, such issues are well understood. The standard machinery of sample complexity (using VC theory) tells us that if the data are *separable* – that is, if they can be perfectly classified by some hypothesis in $\mathcal{H}$ – then we need approximately $1/\epsilon$ random labeled 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.



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

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.

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 $i(h) = \min\{$positive

Pool-based active learning

```
Get a set of unlabeled points $U \subset \mathcal{X}$
Repeat until satisfied:
  Pick some $x \in U$ to label
Return a hypothesis $h \in \mathcal{H}$
```

Stream-based active learning

```
Repeat for $t = 0, 1, 2, \ldots$:
  Choose a hypothesis $h_t \in \mathcal{H}$
  Receive an unlabeled point $x \in \mathcal{X}$
  Decide whether to query its label
```

Active Learning Theory, Fig. 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 $\mathrm{err}_{\mathbb{P}}(h)$. If we want to get this error below ϵ, how many labels are needed, as a function of ϵ?

mass of h, 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 $O(\log 1/\epsilon)$, 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, $\mathrm{err}_{\mathbb{P}}(h)$

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 $\mathrm{err}_{\mathbb{P}}(h) \leq \epsilon$ 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 et al. 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 $\tilde{\Theta}(\min\{1/\mathbb{P}_X([a,b]), 1/\epsilon\} + \log 1/\epsilon)$ when the target hypothesis is $h_{a,b}$ (Dasgupta 2005). Here the $\tilde{\Theta}$ 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 $\tilde{\Theta}(d \log 1/\epsilon)$, exponentially smaller than the $\tilde{O}(d/\epsilon)$ label complexity of supervised learning. If $\mathbb{P}_X$ is not the uniform distribution but is everywhere within a multiplicative factor $\lambda > 1$ of it, then the label complexity becomes $\tilde{O}((d \log 1/\epsilon) \log^2 \lambda)$, 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 et al. 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 et al. 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 et al. 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 et al. 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 et al. 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 et al. 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 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, Washington, DC, pp 12–31
Balcan M-F, Beygelzimer A, Langford J (2006) Agnostic active learning. In: International conference on machine learning. ACM Press, New York, pp 65–72
Balcan M-F, Broder A, Zhang T (2007) Margin based active learning. In: Conference on learning theory, San Diego, pp 35–50
Baum EB, Lang K (1992) Query learning can work poorly when a human oracle is used. In: International joint conference on neural networks, Baltimore
Beygelzimer A, Dasgupta S, Langford J (2009) Importance weighted active learning. In: International conference on machine learning. ACM Press, New York, pp 49–56
Cesa-Bianchi N, Gentile C, Zaniboni L (2004) Worst-case analysis of selective sampling for linear-threshold algorithms. In: Advances in neural information processing systems
Chernoff H (1972) Sequential analysis and optimal design. CBMS-NSF regional conference series in applied mathematics, vol 8. SIAM, Philadelphia
Cohn D, Atlas L, Ladner R (1994) Improving generalization with active learning. Mach Learn 15(2):201–221
Dasgupta S (2005) Coarse sample complexity bounds for active learning. Advances in neural information processing systems. Morgan Kaufmann, San Mateo
Dasgupta S, Kalai A, Monteleoni C (2005) Analysis of perceptron-based active learning. In: 18th annual conference on learning theory, Bertinoro, pp 249–263
Dasgupta S, Hsu DJ, Monteleoni C (2007) A general agnostic active learning algorithm. Advances in neural information processing systems
Fedorov VV (1972) Theory of optimal experiments (trans: Studden WJ, Klimko EM). Academic Press, New York
Freund Y, Seung S, Shamir E, Tishby N (1997) Selective sampling using the query by committee algorithm. Mach Learn J 28:133–168
Friedman E (2009) Active learning for smooth problems. In: Conference on learning theory, Montreal, 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, San Diego, pp 66–81
Hanneke S (2007b) A bound on the label complexity of agnostic active learning. In: International conference on machine learning, Corvallis, pp 353–360
Haussler D (1992) Decision-theoretic generalizations of the PAC model for neural net and other learning applications. Inf Comput 100(1):78–150
Seung HS, Opper M, Sompolinsky H (1992) Query by committee. In: Conference on computational learning theory, Victoria, 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 Learning

▸ Metalearning

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 et al. (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 Asynchronous 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:

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

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 $p_{xy}(a)$ 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.

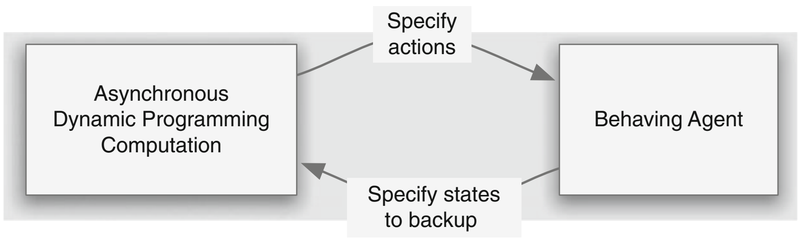
Conventional DP algorithms consist of successive "sweeps" of the state set. Each sweep consists of applying a backup operation to each state. Sweeps continue until the algorithm converges to a solution. Asynchronous DP, which underlies RTDP and ARTDP, does not use systematic sweeps. States can be chosen 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. Bonet and Geffner's (2003a) Labeled RTDP labels states that have already been "solved," allowing faster convergence than RTDP. Feng et al. (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

Recommended Reading

Barto A, Bradtke S, Singh S (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1–2):81–138

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

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

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

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

Hansen E, Zilberstein S (2001) LAO*: a heuristic search algorithm that finds solutions with loops. Artif Intell 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, Tampa, pp 1283–1288

Korf R (1990) Real-time heuristic search. Artif Intell 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). AAAI Press, Boston

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. Morgan Kaufmann, San Mateo, pp 216–224

Adaptive Resonance Theory

Gail A. Carpenter[1] and Stephen Grossberg[2]
[1]Department of Mathematics & Center for Adaptive Systems, Boston University, Boston, MA, USA
[2]Center for Adaptive Systems, Graduate Program in Cognitive and Neural Systems, Department of Mathematics, Boston University, Boston, MA, USA

Abstract

Computational models based on cognitive and neural systems are now deeply embedded in the standard repertoire of machine learning and data mining methods, with intelligent learning systems enhancing performance in nearly every existing application area. Beyond data mining, this article shows how models based on adaptive resonance theory (ART) may provide entirely new questions and practical solutions for technological applications. ART models carry out hypothesis testing, search, and incremental fast or slow, self-stabilizing learning, recognition, and prediction in response to large nonstationary databases (big data). Three computational examples, each based on the distributed ART neural network, frame questions and illustrate how a learning system (each with no free parameters) may enhance the analysis of large-scale data. Performance of each task is simulated on a common mapping platform, a remote sensing dataset called the Boston Testbed, available online along with open-source system code. Key design elements of ART models and links to software for each system are included. The article further points to future applications for integrative ART-based systems that have already been computationally specified and simulated. New application directions include autonomous robotics, general-purpose machine vision, audition, speech recognition, language acquisition, eye movement control, visual search, figure-ground separation, invariant object recognition, social cognition, object and spatial attention, scene understanding, space-time integration, episodic memory, navigation, object tracking, system-level analysis of mental disorders, and machine consciousness.

Adaptive Resonance Theory

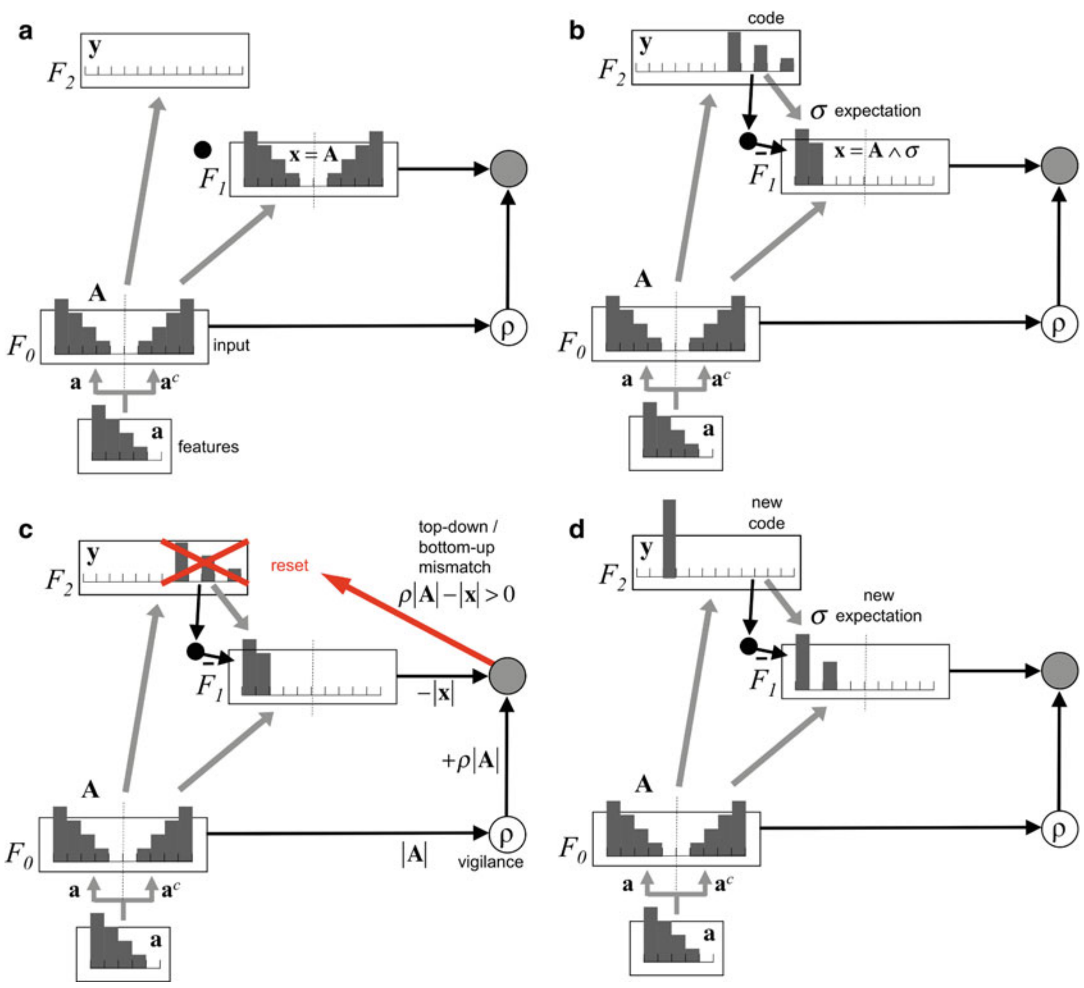
Adaptive resonance theory (ART) neural networks model real-time hypothesis testing, search, learning, recognition, and prediction. Since the 1980s, these models of human cognitive information processing have served as computational engines for a variety of neuromorphic technologies (http://techlab.bu.edu/resources/articles/C5). This article points to a broader range of technology transfers that bring new methods to new problem domains. It describes applications of three specific systems, ART knowledge discovery, self-supervised ART, and biased ART, and summarizes future application areas for large-scale, brain-based model systems.

ART Design Elements

In this article, *ART* refers generally to a theory of cognitive information processing and to an inclusive family of neural models. Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of variants of the basic systems.

Stable Fast Learning with Distributed and Winner-Take-All Coding

ART systems permit fast online learning, whereby long-term memories reach their asymptotes on each input trial. With slow learning, memories change only slightly on each trial. One characteristic that distinguishes classes of ART systems from one another is the nature of their patterns of persistent activation at the coding field F_2 (Fig. 1). The coding field is functionally analogous to the hidden layer of multilayer perceptrons (*Encyclopedia* cross reference). At the perceptron hidden layer, activation is distributed across many nodes, learning needs



Adaptive Resonance Theory, Fig. 1 Distributed ART (dART) (Carpenter 1997). (**a**) At the field F_0, complement coding transforms the feature pattern **a** to the system input **A**, which represents both scaled feature values $a_i \in [0,1]$ and their complements $(1-a_i)$ $(i=1\ldots M)$. (**b**) F_2 is a competitive field that transforms its input pattern into the working memory code **y**. The F_2 nodes that remain active following competition send the pattern σ of learned top-down expectations to the match field F_1. The pattern active at F_1 becomes $\mathbf{x} = \mathbf{A} \wedge \sigma$, where $\wedge$ denotes the component-wise minimum, or fuzzy intersection. (**c**) A parameter $\rho \in [0,1]$, called vigilance, sets the matching criterion. The system registers a mismatch if the size of **x** is less than ρ times the size of **A**. A top-down/bottom-up mismatch triggers a signal that resets the active F_2 code. (**d**) Medium-term memories in the F_0-to-F_2 dynamic weights allow the system to activate a new code **y**. When only one F_2 node remains active following competition, the code is maximally compressed, or winner-take-all. When $|\mathbf{x}| \geq \rho|\mathbf{A}|$, the activation pattern **y** persists until the next reset, even if input **A** changes or F_0-to-F_2 signals habituate. During learning, thresholds τ_{ij} in paths from F_0 to F_2 increase according to the dInstar law; and thresholds τ_{ji} in paths from F_2 to F_1 increase according to the dOutstar law

to be slow, and activation does not persist once inputs are removed. The ART coding field is a competitive network where, typically, one or a few nodes in the normalized F_2 pattern **y** sustain persistent activation, even as their generating inputs shift, habituate, or vanish. The pattern **y** persists until an active reset signal (Fig. 1c) prepares the coding field to register a new F_0-to-F_2 input. Early ART networks (Carpenter and Grossberg 1987; Carpenter et al. 1991a, 1992) employed *localist*, or *winner-take-all*, coding, whereby strongly competitive feedback results in only one F_2 node staying active until the next reset. With fast as well as slow learning, memory stability in these early networks relied on their winner-take-all architectures.

Achieving stable fast learning with distributed code representations presents a computational challenge to any learning network. In order to meet this challenge, distributed ART (Carpenter 1997) introduced a new network configuration (Fig. 1) in which system fields are identified with cortical layers (Carpenter 2001). New learning laws (*dInstar* and *dOutstar*) that realize stable fast learning with distributed coding predict adaptive dynamics between cortical layers.

Distributed ART (dART) systems employ a new unit of long-term memory, which replaces the traditional multiplicative weight (*Encyclopedia* cross reference) with a *dynamic weight* (Carpenter 1994). In a path from the F_2 coding node j to the F_1 matching node i, the dynamic weight equals the amount by which coding node activation y_j exceeds an *adaptive threshold* τ_{ji}. The total signal σ_i from F_2 to the i^{th} F_1 node is the sum of these dynamic weights, and F_1 node activation x_i equals the minimum of the top-down expectation σ_i and the bottom-up input A_i. During dOutstar learning, the top-down pattern σ converges toward the matched pattern $\mathbf{x}$.

When coding node activation y_j is below τ_{ji}, the dynamic weight is zero and no learning occurs in that path, even if y_j is positive. This property is critical for stable fast learning with distributed codes. Although the dInstar and dOutstar laws are compatible with F_2 patterns $\mathbf{y}$ that are arbitrarily distributed, in practice, following an initial learning phase, most changes in paths to and from a coding node j occur only when its activation y_j is large. This type of learning is therefore called *quasi-localist*. In the special case where coding is winner-take-all, the dynamic weight is equivalent to a multiplicative weight that formally equals the *complement* of the adaptive threshold.

Complement Coding: Learning Both Absent Features and Present Features

ART networks employ a preprocessing step called *complement coding* (Carpenter et al. 1991b), which models the nervous system's ubiquitous computational design known as *opponent processing* (Hurvich and Jameson 1957). Balancing an entity against its opponent, as in opponent colors such as red vs. green or agonist-antagonist muscle pairs, allows a system to act upon relative quantities, even as absolute magnitudes fluctuate unpredictably. In ART systems, complement coding is analogous to retinal on-cells and off-cells (Schiller 1982). When the learning system is presented with a set of input features $\mathbf{a} \equiv (a_1 \ldots a_i \ldots a_M)$, complement coding doubles the number of input components, presenting to the network an input $\mathbf{A}$ that concatenates the original feature vector and its complement (Fig. 1a).

Complement coding produces normalized inputs $\mathbf{A}$ that allow a model to encode 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 F_2 node is highly active are regarded as uninformative with respect to that node, and the corresponding *present* and *absent* top-down feature expectations shrink to zero. When a new input activates this node, these features are suppressed at the match field F_1 (Fig. 1b). If the active code then produces an error signal, attentional biasing can enhance the salience of input features that it had previously ignored, as described below.

Matching, Attention, and Search

A neural computation central to both scientific and technological analyses is the *ART matching rule* (Carpenter and Grossberg 1987), which controls how attention is focused on critical feature patterns via dynamic matching of a bottom-up sensory input with a top-down learned expectation. Bottom-up/top-down pattern matching and attentional focusing are, perhaps, the primary features common to all ART models across their many variations. Active input features that are not confirmed by top-down expectations are inhibited (Fig. 1b). The remaining activation pattern defines a focus of attention, which, in turn, determines what feature patterns are learned. Basing memories on attended features rather than whole patterns supports the design goal of encoding stable memories with fast as well as slow learning. Encoding attended feature subsets also enables one-to-many learning, where the system may attach many context-dependent labels (*Spot*, *dog*, *animal*) to one input. This capability promotes knowledge discovery ($Spot \Rightarrow dog$ and $dog \Rightarrow animal$) in a learning system that experiences one input at a time, with no explicit connection between inputs.

When the match is good enough, F_2 activation persists and learning proceeds. Where they exceed the corresponding bottom-up input components, top-down signals decay as expectations converge toward the attended pattern at F_1. The coding field F_2 contains a reserve of *uncommitted* coding nodes, which compete with the previously active *committed* nodes. When a previously uncommitted node is first activated during supervised learning, it is associated with its designated output class. During testing, the selection of an uncommitted node means *I don't know*. ART networks for supervised learning are called *ARTMAP* (Carpenter et al. 1991a, 1992).

A mismatch between an active top-down expectation and the bottom-up input leads to a parallel memory search (Fig. 1c). The ART matching criterion is set by a *vigilance parameter* ρ. Low vigilance permits the learning of broad classes, across diverse exemplars, while high vigilance limits learning to narrow classes. When a new input arrives, vigilance equals a baseline level. Baseline vigilance is set equal to zero to maximize generalization. ARTMAP vigilance increases following a predictive error or negative reinforcement (*Encyclopedia* cross reference). The internal computation that determines how far ρ rises to correct the error is called *match tracking* (Carpenter et al. 1991a). As vigilance rises, the network pays more attention to how well top-down expectations match the bottom-up input. The match tracking modification MT– (Carpenter and Markuzon 1998) also allows the system to learn inconsistent cases. For example, three similar, even identical, map regions may have been correctly labeled by different observers as *ocean* or *water* or *natural*. The ability to learn one-to-many maps, which can label a single test input as *ocean* and *water* and *natural*, is a key feature of the ART knowledge discovery system described below.

Applications

Three computational examples illustrate how cognitive and neural systems can introduce new approaches to the analysis of large datasets. Application 1 (self-supervised ART) addresses the question: how can a neural system learning from one example at a time absorb information that is inconsistent but correct, as when a family pet is called *Spot* and *dog* and *animal*, while rejecting similar incorrect information, as when the same pet is called *wolf*? How does this system transform scattered information into knowledge that *dogs are animals*, but not conversely? Application 2 (ART knowledge discovery) asks: how can a real-time system, initially trained with a few labeled examples and a limited feature set, continue to learn from experience, without supervision, when confronted with oceans of additional information, without eroding reliable early memories? How can such individual systems adapt to their unique application contexts? Application 3 (biased ART) asks: how can a neural system that has made an error refocus attention on features that it initially ignored?

The Boston Testbed

The Boston Testbed was developed to compare performance of learning systems applied to challenging problems of spatial analysis. Each multispectral Boston image pixel produces 41 feature values: 6 Landsat 7 Thematic Mapper (TM) bands at 30 m resolution, 2 thermal bands at 60 m resolution, 1 panchromatic band at 15 m resolution, and 32 derived bands representing local contrast, color, and texture. In the Boston dataset, each of 28,735 ground truth pixels is labeled as belonging to one of seven classes (*beach, ocean, ice, river, park, residential, industrial*). For knowledge discovery system training, some *ocean, ice*, and *river* pixels are instead labeled as belonging to broader classes such as *water* or *natural*. No pixel has more than one label, and the learning system is given no information about relationships between target classes. The labeled dataset is available from the CNS Technology Lab Website [http://techlab.bu.edu/classer/data_sets/].

A cross-validation procedure divides an image into four vertical strips: two for training, one for validation (if needed for parameter selection), and one for testing. Class mixtures differ markedly across strips. For example, one strip contains many *ocean* pixels, while another strip contains neither *ocean* nor *beach* pixels. Geographically dissimilar training and testing areas robustly assess regional generalization. In this article, spatial analysis simulations on the Boston Testbed follow this protocol to illustrate ART systems for self-supervised learning, knowledge discovery, and attentional control. Since each system in Applications 1–3 requires no parameter selection, training uses randomly chosen pixels from three strips, with testing on the fourth strip.

Application 1: Learning from Experience with Self-Supervised ART

Computational models of supervised pattern recognition typically utilize two learning phases. During an initial training phase, input patterns, described as specified values of a set of features, are presented along with output class labels or patterns. During a subsequent testing phase, the model generates output predictions for unlabeled inputs, and no further learning takes place.

Although supervised learning has been successfully applied in diverse settings, it does not reflect many natural learning situations. Humans do learn from explicit training, as from a textbook or a teacher, and they do take tests. However, students do not stop learning when they leave the classroom. Rather, they continue to learn from experience, incorporating not only more information but new types of information, all the while building on the foundation of their earlier knowledge. Self-supervised ART models such life-long learning.

An unsupervised learning system clusters unlabeled input patterns. *Semi-supervised* learning incorporates both labeled and unlabeled inputs in its training set, but all inputs typically have the same number of specified feature values. Without any novel features from which to learn, semi-supervised learning systems use unlabeled data to refine the model parameters defined using labeled data. Reviews of semi-supervised learning (Chapelle et al. 2006) have found that many of the successful models are carefully selected and tuned, using a priori knowledge of the problem. Chapelle et al. (2006) conclude that none of the semi-supervised models they review is robust enough to be general purpose. The main difficulty seems to be that, whenever unlabeled instances are different enough from labeled instances to merit learning, these differences could contain misinformation that may damage system performance.

The *self-supervised* paradigm models two learning stages. During Stage 1 learning, the system receives all output labels, but only a subset of possible feature values for each input. During Stage 2 learning, the system may receive more feature values for each input, but no output labels. In Stage 1, when the system can confidently incorporate externally specified output labels, self-supervised ART (Amis and Carpenter 2010) employs winner-take-all coding and fast learning. In Stage 2, when the system internally generates its own output labels, codes are distributed so that incorrect hypotheses do not abruptly override reliable "classroom learning" of Stage 1. The distributed ART learning laws, dInstar (Carpenter 1997) and dOutstar (Carpenter 1994), scale memory changes to internally generated measures of prediction confidence and prevent memory changes altogether for most inputs. Memory stability derives from the dynamic weight representation of long-term memories, which permits learning only in paths to and from highly active coding nodes. Dynamic weights solve a problem inherent in learning laws based on multiplicative weights, which are prone to catastrophic forgetting when implemented with distributed codes and huge datasets, even when learning is very slow.

In addition to emulating the human learning experience, self-supervised learning maps to technological applications that need to cope with huge, ever-changing datasets. A supervised learning system that completes all training before making test predictions does not adapt to new information and individual contexts. A semi-supervised system risks degrading its supervised knowledge. Self-supervised ART continues to learn from new experiences, with built-in safeguards that conserve useful memories. Self-supervised ART code is available from the CNS Technology Lab Website (http://techlab.bu.edu/SSART/).

A simulation study based on the Boston Testbed (Amis and Carpenter 2010) illustrates ways in which high-dimensional problems may challenge *any system* learning without labels. As in most ground truth datasets, labeled pixels consist primarily of clear exemplars of single classes. Because sensors have a 15–60 m resolution, many unlabeled pixels cover multiple classes, such as *ice* and *industrial*. Stage 2 inputs thus mix and distort features from multiple classes, placing many of the unlabeled feature vectors far from the distinct class clusters of the Stage 1 training set. Although the distributed ART learning laws are open to unrestricted adaptation on any pixel, the distributed codes of Stage 2 minimize the influence of mixed pixels. Most memory changes occur on unambiguous cases, despite the fact that the unlabeled pixels provide no external indices of class ambiguity. Self-supervised Stage 2 learning dramatically improves performance compared to learning that ends after Stage 1. On every one of 500 individual simulations, Stage 2 learning improves test accuracy, as unlabeled fully featured inputs consistently expand knowledge from Stage 1 training.

Application 2: Transforming Information into Knowledge Using ART Knowledge Discovery

Classifying terrain or objects may require the resolution of conflicting information from sensors working at different times, locations, and scales and from users with different goals and situations. *Image fusion* has been defined as "the acquisition, processing and synergistic combination of information provided by various sensors or by the same sensor in many measuring contexts" (Simone et al. 2002, p. 3). When multiple sources provide inconsistent data, fusion methods are called upon to appraise information components to decide among various options and to resolve inconsistencies, as when evidence suggests that an object is a *car* or a *truck* or a *bus*. Fusion methods weigh the confidence and reliability of each source, merging complementary information or gathering more data. In any case, at most one of these answers is correct.

The method described here defines a complementary approach to the information fusion problem, considering the case where sensors and sources are both nominally inconsistent and reliable, as when evidence suggests that an object is a *car* and a *vehicle* and *man-made* or when a *car* is alternatively labeled *automobile*. Underlying relationships among classes are assumed to be unknown to the automated system or the human user, as if the labels were encrypted.

The ART knowledge discovery model acts as a self-organizing expert system to derive consistent knowledge structures from such nominally inconsistent data (Carpenter et al. 2005). Once derived, a rule set can be used to assign classes to levels. For each rule $x \Rightarrow y$, class x is located at a lower level than class y. Classes connected by arrows that codify a list of rules and confidence values form a graphical representation of a knowledge hierarchy. For spatial data, the resulting diagram of the relationships among classes can guide the construction of orderly layered maps. ART knowledge discovery code is available from the CNS Technology Lab Website (http://techlab.bu.edu/classer/classer_toolkit_overview). On the Boston Testbed, the ART knowledge discovery system places each class at its correct level and finds all the correct rules for this example.

Application 3: Correcting Errors by Biasing Attention Using Biased ART

Memories in ART networks are based on matched patterns that focus attention on *critical features*, where bottom-up inputs match active top-down expectations. While this learning strategy has proved successful for both brain models and applications, computational examples demonstrate that paying too much attention to critical features that have been selected to represent a given category early on may distort memory representations during subsequent learning. If training inputs are repeatedly presented, an ART system will correct these initial errors. However, real-time learning may not afford such repeat opportunities. Biased ART (bART) (Carpenter and Gaddam 2010) solves the problem of overemphasis on early critical features by directing attention away from initially attended features after the system makes a predictive error.

Activity **x** at the ART field F_1 computes the match between the field's bottom-up and top-down input patterns (Fig. 1). A reset signal shuts off the active F_2 code when **x** fails to meet the matching criterion determined by vigilance ρ. Reset alone does not, however, induce a different code: unless the prior code has left an enduring trace within the F_0–F_2 subsystem, the network will simply reactivate the same pattern at F_2.

Following reset, all ART systems shift attention away from previously active *coding* nodes at the field F_2. As modeled in ART 3 (Carpenter and Grossberg 1990), biasing the bottom-up input to the coding field to favor previously inactive F_2 nodes implements search by enabling the network to activate a new code in response to a reset signal. The ART 3 search mechanism defines a medium-term memory in the F_0-to-F_2 adaptive filter so that the system does not perseverate indefinitely on an output class that had just produced a reset. A presynaptic interpretation of this bias mechanism is transmitter depletion or habituation.

The biased ART network (Carpenter and Gaddam 2010) introduces a second, top-down, medium-term memory which, following reset, shifts attention away from previously active *feature* nodes at the match field F_1. In Fig. 1, the first feature is strongly represented in the input **A** and in the matched patterns **x** at F_1 both before reset (Fig. 1b) and after reset (Fig. 1d). Following the same sequence as in Fig. 1a–c, biased ART would diminish the size of the first feature in the matched pattern. The addition of featural biasing helps the system to pay more attention to input features that it had previously ignored.

The biasing mechanism is a small modular element that can be added to any ART network. While computational examples and Boston Testbed simulations demonstrate how featural biasing in response to predictive errors improves performance on supervised learning tasks, the error signal that gates biasing could have originated from other sources, as in reinforcement learning. Biased ART code is available from the CNS Technology Lab Website (http://techlab.bu.edu/bART).

Future Directions

Applications for tested software based on computational intelligence abound. This section outlines areas where ART systems may open qualitatively new frontiers for novel technologies. Future applications summarized here would adapt and specialize brain models that have already been mathematically specified and computationally simulated to explain and predict large psychological and neurobiological databases. By linking the brain to mind, these models characterize both mechanism (how the model works) and function (what the model is for). Both mechanism and function are needed to design new applications. These systems embody new designs for autonomous adaptive agents, including new computational paradigms that are called Complementary Computing and Laminar Computing. These paradigms enable the autonomous adaptation in real time of individual persons or machines to nonstationary situations filled with unexpected events. See Grossberg (2013) for a review.

New Paradigms for Autonomous Intelligent Systems: Complementary Computing and Laminar Computing

Functional integration is essential to the design of a complex autonomous system such as a robot moving and learning freely in an unpredictable environment. Linking independent modules for, say, vision and motor control will not necessarily produce a coordinated system that can adapt to unexpected events in changeable contexts. How, then, should such an autonomous adaptive system be designed?

A clue can be found in the nature of brain specialization. How have brains evolved while interacting with the physical world and embodying its invariants? Many scientists have proposed that our brains possess independent modules. The brain's organization into distinct anatomical areas and processing streams shows that brain regions are indeed specialized. Whereas independent modules compute their particular processes on their own, behavioral data argue against this possibility. *Complementary Computing* (Grossberg 2000a,b, 2013) concerns the discovery that pairs of parallel cortical processing streams compute computationally complementary properties. Each stream has complementary strengths and weaknesses, much as in physical principles like the Heisenberg uncertainty principle. Each cortical stream can also possess multiple processing stages. These stages realize a *hierarchical resolution of uncertainty*. "Uncertainty" here means that computing one set of properties at a given stage prevents computation of a complementary set of properties at that stage. Complementary Computing proposes that the computational unit of brain processing that has behavioral significance consists of parallel and hierarchical interactions between complementary cortical processing streams with multiple processing stages. These interactions overcome complementary weaknesses to compute necessary information about a particular type of biological intelligence.

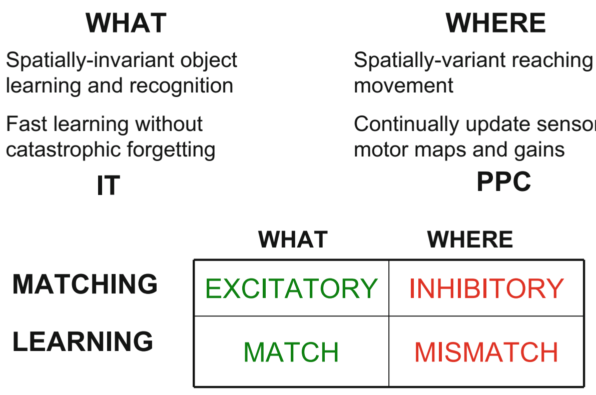
Five decades of neural modeling have shown how Complementary Computing is embedded as a fundamental design principle in neural systems for vision, speech and language, cognition, emotion, and sensory-motor control. Complementary Computing hereby provides a blueprint for designing large-scale autonomous adaptive systems that are poised for technological implementation.

A unifying anatomical theme that enables communication among cortical systems is *Laminar Computing*. The cerebral cortex, the seat of higher intelligence in all modalities, is organized into layered circuits (often six main layers) that undergo characteristic bottom-up, top-down, and horizontal interactions. As information travels up and down connected regions, distributed decisions are made in real time based on a preponderance of evidence. Multiple levels suppress weaker groupings while communicating locally coherent choices. The distributed ART model (Fig. 1), for example, features three cortical layers, with its distributed code (e.g., at a cortical layer 6) producing a distributed output. Stacks of match fields (inflow) and coding fields (outflow) lay the substrate for cortical hierarchies.

How do specializations of this shared laminar design embody different types of biological intelligence, including vision, speech, language, and cognition? How does this shared design enable seamless intercortical interactions? Models of Laminar Computing clarify how these different types of intelligence all use variations of the same laminar circuitry (Grossberg 2013; Grossberg and Pearson 2008). This circuitry represents a revolutionary synthesis of desirable computational properties of feedforward and feedback processing, digital and analog processing, and bottom-up data-driven processing and top-down attentive hypothesis-driven processing. Realizing such designs in hardware that embodies biological intelligence promises to facilitate the development of increasingly general-purpose adaptive autonomous systems for multiple applications.

Complementary Computing in the Design of Perceptual/Cognitive and Spatial/Motor Systems

Many neural models that embody subsystems of an autonomous adaptive agent have been developed and computationally characterized. It remains to unify and adapt them to particular machine learning applications. Complementary Computing implies that not all of these subsystems could be based on variants of ART. In particular, accumulating experimental and theoretical evidence shows that perceptual/cognitive and spatial/motor processes use different learning, matching, and predictive laws for their complementary functions (Fig. 2). ART-like processing is ubiquitous in perceptual and cognitive processes, including excitatory matching and match-based learning that enables self-stabilizing memories to form. Vector


WHAT
Spatially-invariant object learning and recognition
Fast learning without catastrophic forgetting
IT
WHERE
Spatially-variant reaching and movement
Continually update sensory-motor maps and gains
PPC
WHAT
WHERE
MATCHING
LEARNING
EXCITATORY
INHIBITORY
MATCH
MISMATCH

Adaptive Resonance Theory, Fig. 2 Complementary What and Where cortical processing streams for spatially invariant object recognition and spatially variant spatial representation and action, respectively. Perception and recognition use top-down excitatory matching and match-based fast or slow learning without catastrophic forgetting. Spatial and motor tasks use inhibitory matching and mismatch-based learning to achieve adaptation to changing bodily parameters. *IT* inferotemporal cortex, *PPC* posterior parietal cortex

Associative Map (VAM) processing is often found in spatial and motor processes, which rely on inhibitory matching and mismatch-based learning. In these modalities, spatial maps and motor plants are adaptively updated without needing to remember past maps and parameters. Complementary mechanisms create a self-stabilizing perceptual/cognitive front end for intelligently manipulating the more labile spatial/motor processes that enable our changeable bodies to act effectively upon a changing world.

Some of the existing large-scale ART systems are briefly reviewed here, using visually based systems for definiteness. Citations refer to articles that specify system equations and simulations and that can be downloaded from http://cns.bu.edu/~steve.

Where's Waldo? Unifying Spatial and Object Attention, Learning, Recognition, and Search of Valued Objects and Scenes

ART models have been incorporated into larger system architectures that clarify how individuals autonomously carry out intelligent behaviors as they explore novel environments. One such development is the ARTSCAN family of architectures, which model how individuals rapidly learn to search a scene to detect, attend, invariantly recognize, and look at a valued target object (Fig. 3; Cao, Grossberg, and Markowitz 2011; Chang, Grossberg, and Cao 2014; Fazl, Grossberg, and Mingolla 2009; Foley, Grossberg, and Mingolla 2012; Grossberg, Srinivasan, and Yazdanbakhsh 2014). Such a competence represents a proposed solution of the Where's Waldo problem.

The ventral What stream is associated with object learning, recognition, and prediction, whereas the dorsal Where stream carries out processes such as object localization, spatial attention, and eye movement control. To achieve efficient object recognition, the What stream learns object category representations that become increasingly invariant under view, size, and position changes at higher processing stages. Such invariance enables objects to be learned and recognized without causing a combinatorial explosion. However, by stripping away the positional coordinates of each object exemplar, the What stream loses the ability to command actions to the positions of valued objects. The Where stream computes positional representations of the world and controls actions to acquire objects in it, but does not represent detailed properties of the objects themselves.

ARTSCAN architectures model how an autonomous agent can determine when the views that are foveated by successive scanning movements belong to the same object and thus determine which view-selective categories should be associatively linked to an emerging view- , size-, and positionally-invariant object category. This competence, which avoids the problem of erroneously merging pieces of different objects, works even under the unsupervised learning conditions that are the norm during many object learning experiences in vivo. The model identifies a new role for spatial attention in the Where stream, namely, control of invariant object category learning by the What stream. Interactions across the What and Where streams overcome the deficiencies of computationally complementary properties of these streams.

In the ARTSCAN Search model, both Where-to-What and What-to-Where stream interactions are needed to overcome complementary weaknesses: Where stream processes of spatial attention and predictive eye movement control regulate What stream processes whereby multiple view- and positionally-specific object categories are learned and associatively linked to view- and positionally-invariant object categories through bottom-up and object-attentive top-down interactions. What stream cognitive-emotional learning processes enable the focusing of motivated attention upon the invariant object categories of desired objects (Brown, Bullock, and Grossberg 1999, 2004; Dranias, Grossberg, and Bullock 2008; Grossberg and Seidman 2006). What stream cognitive names or motivational drives can, together with volitional signals, drive a search for Waldo. Mediated by object attention, search proceeds from What stream positionally-invariant representations to Where stream positionally-specific representations that focus spatial attention on Waldo's position. ARTSCAN architectures hereby model how the dynamics of multiple brain regions are coordinated to achieve clear functional goals.

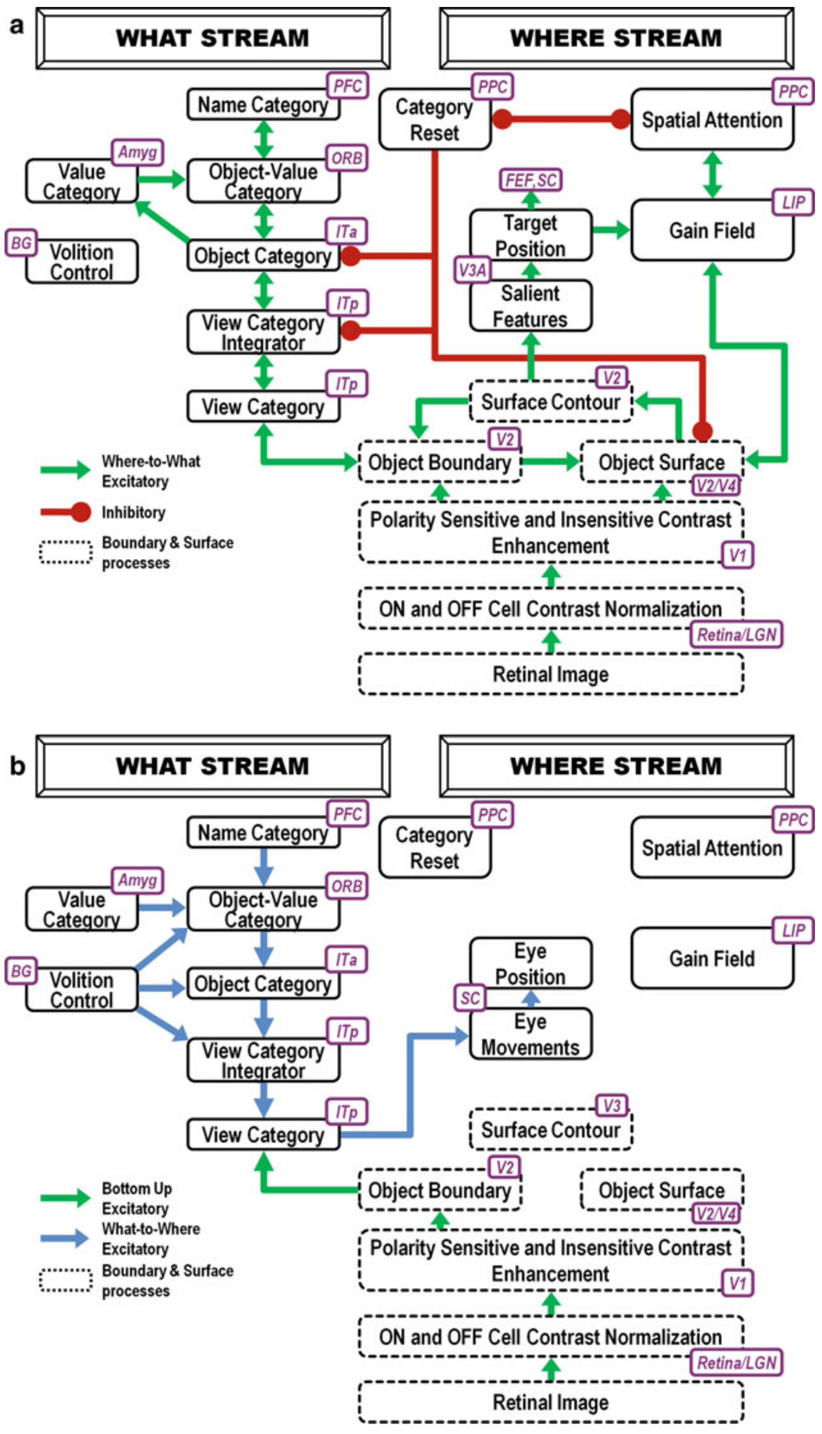
The focus of spatial attention on Waldo's position in the Where stream can be used to control eye and hand movements toward Waldo, after navigational circuits (see below) bring the observer close enough to contact him. VAM-type learning circuits have been developed for the control of goal-oriented eye and hand movements that can be used for this purpose (e.g., Bullock and Grossberg 1988, 1991; Bullock, Cisek, and Grossberg 1998; Contreras-Vidal, Grossberg, and Bullock 1997; Gancarz and Grossberg 1999; Grossberg, Srihasam, and Bullock 2012; Pack, Grossberg, and Mingolla 2001; Srihasam, Bullock, and Grossberg 2009).

The ARTSCENE system (Grossberg and Huang 2009) models how humans can incrementally learn and rapidly predict scene identity by gist and then accumulates learned evidence from scenic textures to refine its initial hypothesis, using the same kind of spatial attentional shrouds that help to learn invariant object categories in ARTSCAN. The ARTSCENE Search system (Huang and Grossberg 2010) models how humans use target-predictive contextual information to guide search for desired targets in familiar scenes. For example, humans can learn that a certain combination of objects may define a context for a kitchen and trigger a more efficient search for a typical object, such as a sink, in that context.

General-Purpose Vision and How It Supports Object Learning, Recognition, and Tracking

Visual preprocessing constrains the quality of visually based learning and recognition. On an assembly line, automated vision systems successfully scan for target objects in this carefully controlled environment. In contrast, a human or robot navigating a natural scene faces overlaid textures, edges, shading, and depth information, with multiple scales and shifting perspectives. In the human brain, evolution has produced a huge preprocessor, involving multiple brain regions, for object and scene representation and for target tracking and navigation. One reason for this is that visual boundaries and surfaces, visual form and motion, and target tracking and visually based navigation are computationally complementary, thus requiring several distinct but interacting cortical processing streams.

Prior to the development of systems such as ARTSCAN and ARTSCENE, the FACADE (Form-And-Color-And-DEpth) model provided



Adaptive Resonance Theory, Fig. 3 (continued) a neural theory of form perception, including 3D vision and figure-ground separation (e.g., Cao and Grossberg 2005, 2012; Fang and Grossberg 2009; Grossberg, Kuhlmann, and Mingolla 2007; Grossberg and Swaminathan 2004; Kelly and Grossberg 2000). The 3D FORMOTION model provides a neural theory of motion processing and form-motion interactions (e.g., Baloch and Grossberg 1997; Baloch, Grossberg, Mingolla, and Nogueira 1999; Berzhanskaya, Grossberg, and Mingolla 2007; Grossberg, Leveille, and Versace 2011; Grossberg, Mingolla, and Viswanathan 2001; Grossberg and Rudd 1992). The FACADE model has just the properties that are needed for solving the Where's Waldo problem, and the 3D FORMOTION model has just the properties that are needed for tracking unpredictably moving targets. Their complementary properties enabled these extensions.

Visual and Spatial Navigation, Cognitive Working Memory, and Planning

In addition to being able to see, learn, recognize, and track valued goal objects, an animal or autonomous robot must also be able to navigate to or away from them and to interact with them through goal-oriented hand and arm movements. Navigation is controlled by two distinct and interacting systems: a visually guided system and a spatial path integration system.

Visually guided navigation through a cluttered natural scene is modeled using the 3D FORMOTION model as a front end. The STARS and ViSTARS neural systems (Browning, Grossberg, and Mingolla 2009a,b; Elder, Grossberg, and Mingolla 2009) model how primates use object motion information to segment objects and optic flow information to determine heading (self-motion direction), for purposes of goal approach and obstacle avoidance in response to realistic environments. The models predict how computationally complementary processes in parallel streams within the visual cortex compute object motion for tracking and self-motion for navigation. The models' steering decisions compute goals as attractors and obstacles as repellers, as do humans.

Spatial navigation based upon path integration signals has been a topic of great interest recently. Indeed, the 2014 Nobel Prize in Physiology or Medicine was awarded to John O'Keefe for his discovery of place cells in the hippocampal cortex and to Edvard and May-Britt Moser for their discovery of grid cells in the entorhinal cortex. The GridPlaceMap neural system (Grossberg and Pilly 2012, 2014; Pilly and Grossberg 2012, 2014; Mhatre, Grossberg, and Gorchetch-

◄

Adaptive Resonance Theory, Fig. 3 ARTSCAN Search macrocircuit and corresponding brain regions. *Dashed boxes* indicate boundary and surface preprocessing. (**a**) Category learning system. *Arrows* represent excitatory cortical processes. Spatial attention in the Where stream regulates view-specific and view-invariant category learning and recognition, and attendant reinforcement learning, in the What stream. Connections ending in circular disks indicate inhibitory connections. (**b**) Where's Waldo search system. Search begins when a name category or value category is activated and subliminally primes an object-value category via the *ART matching rule*. A volition control signal enables the primed object-value category to fire output signals. Bolstered by volitional control signals, these output signals can, in turn, propagate through a positionally-invariant object category to all the positionally-variant view category integrators whose various views and positions are represented by the object category. The view category integrators can subliminally prime, but not fully activate, these view categories. All this occurs in the What stream. When the bottom-up input from an object's boundary/surface representation also activates one of these view categories, its activity becomes suprathreshold, wins the competition across view categories for persistent activation, and activates a spatial attentional representation of Waldo's position in the Where stream. *ITa* anterior part of inferotemporal cortex, *ITp* posterior part of inferotemporal cortex, *PPC* posterior parietal cortex, *LIP* lateral intraparietal cortex, *LGN* lateral geniculate nucleus, *ORB* orbitofrontal cortex, *Amyg* amygdala, *BG* basal ganglia, *PFC* prefrontal cortex, *SC* superior colliculus, *V1* striate visual cortex, *V2, V3,* and *V4* prestriate visual cortices nikov 2012; Pilly and Grossberg 2014) proposes how entorhinal grid cells and hippocampal place cells may be learned as spatial categories in a hierarchy of self-organizing maps. The model responds to realistic rat navigational trajectories by learning both grid cells with hexagonal grid firing fields of multiple spatial scales, and place cells with one or more firing fields. Model dynamics match neurophysiological data about their development in juvenile rats. The GridPlaceMap model enjoys several parsimonious design features that will facilitate their embodiment in technological applications, including hardware: (1) similar ring attractor mechanisms process both linear and angular path integration inputs that drive map learning; (2) the same self-organizing map mechanisms can learn grid cell and place cell receptive fields in a hierarchy of maps, and both grid and place cells can develop by detecting, learning, and remembering the most frequent and energetic co-occurrences of their inputs; and (3) the learning of the dorsoventral organization of grid cell modules with multiple spatial scales that occur in the pathway from the medial entorhinal cortex to hippocampus seems to use mechanisms that are homologous to those for adaptively timed temporal learning that occur in the pathway from the lateral entorhinal cortex to hippocampus (Grossberg and Merrill 1989, 1992; Grossberg and Schmajuk 1989). The homologous mechanisms for representing space and time in this entorhinal-hippocampal system has led to the phrase "neural relativity" for this parsimonious design.

Finally, the GridPlaceMap model is an ART system. It proposes how top-down hippocampus-to-entorhinal attentional mechanisms may stabilize map learning and thereby simulates how hippocampal inactivation may disrupt grid cell properties and explains challenging data about theta, beta, and gamma oscillations.

Visual and path integration information cooperate during navigation. Cognitive planning also influences navigational decisions. More research is needed to show how learning fuses visual, path integration, and planning circuits into a unified navigational system. The design of a general planning system will be facilitated by the fact that similar circuits for short-term storage of event sequences (working memory) and for learning of sequential plans are used by the brain to control linguistic, spatial, and motor behaviors (Grossberg and Pearson 2008; Silver, Grossberg, Bullock, Histed, and Miller 2011).

Social Cognition

How can multiple autonomous systems interact intelligently? Individuals experience the world from self-centered perspectives. What we learn from each other is thus computed in different coordinates within our minds. How do we bridge these diverse coordinates? A model of social cognition that explains how a teacher can instruct a learner who experiences the world from a different perspective can be used to enable a single human or robotic teacher to instruct a large "class" of embodied robots that all experience the teacher from different perspectives.

Piaget's *circular reaction* notes the feedback loop between the eye and hand in the learning infant, laying the foundation for visually guided reaching. Similarly, feedback between babbled sounds and hearing forms the learned substrate of language production. These *intra*personal circular reactions were extended to *inter*personal circular reactions within the Circular Reactions for Imitative Behavior (CRIB) model (Grossberg and Vladusich 2010). This model shows how social cognition builds upon ARTSCAN mechanisms. These mechanisms clarify how an infant learns how to share joint attention with adult teachers and to follow their gaze toward valued goal objects. The infant also needs to be capable of view-invariant object learning and recognition whereby it can carry out goal-directed behaviors, such as the use of tools, using different object views than the ones that its teachers use. Such capabilities are often attributed to *mirror neurons*. This attribution does not, however, explain the brain processes whereby these competences arise. CRIB proposes how intrapersonal circular reactions create a foundation for interpersonal circular reactions when infants and other learners interact with external teachers in space. Both types of circular reactions involve learned coordinate transformations between body-centered arm movement commands and retinotopic visual feedback, and coordination of processes within and between the What and Where cortical processing streams. Specific breakdowns of model processes generate formal symptoms similar to clinical symptoms of autism.

Mental Disorders and Homeostatic Plasticity

Optimally functioning autonomous intelligent systems require properly balanced complementary systems. What happens when they become imbalanced? In humans, they can experience mental disorders.

Scientific literature on human mental disorders such as autism and schizophrenia is, of necessity, more anecdotal than parametric and is, therefore, an insufficient foundation for model construction. Real-time models of normal mental behavior that are based on the huge databases from decades of psychological and neurobiological experiments have, however, provided insights into the mechanisms of abnormal behaviors (e.g., Carpenter and Grossberg 1993; Grossberg 1984, 2000a,b; Grossberg and Seidman 2006).

Imbalanced processes across the complementary systems that control normal behaviors can produce constellations of model symptoms that strikingly resemble mental disorders. For example, fixing the ART vigilance parameter ρ at too high a level leads to symptoms familiar in autistic individuals, notably learning of hyperconcrete categories and difficulty paying attention to the meaning of a task. Underarousal of the model amygdala can lead to insensitivity to social meanings and also to intense emotional outbursts and coping strategies to reduce event complexity and unexpectedness. Damage to the model cerebellum can lead to defects of adaptively timed learning and thus a host of problems in socialization.

In both humans and robots, it remains an open problem to model how biologically based autonomous systems can discover and maintain their own optimal operating parameters in response to the challenges of an unpredictable world. An initial step toward solving this *homeostatic plasticity* problem was made in Chandler and Grossberg (2012).

Machine Consciousness?

An early ART prediction is that *all conscious states are resonant states,* though not all resonant states are conscious. Since that time, ART has predicted how specific resonances support different kinds of consciousness. These observations suggest the question: can machines that embody ART resonant dynamics experience a type of consciousness? For example, ART models predict that *surface-shroud resonances* subserve conscious percepts of visual qualia, *feature-category resonances* subserve recognition of familiar objects and scenes, *spectral-shroud resonances* subserve conscious percepts of auditory streams, spectral-pitch-and-timbe resonances subserve conscious recognition of auditory streams, *item-list resonances* subserve conscious percepts of speech and language, and *cognitive-emotional resonances* subserve conscious feelings and knowing the objects or events that cause them. ART models also identify the brain regions and interactions that would support these resonances.

These results about model correlates of consciousness emerge from ART analyses of the mechanistic relationships among processes of Consciousness, Learning, Expectation, Attention, Resonance, and Synchrony (the CLEARS processes). Recall, however, that not all resonant states are conscious states. For example, entorhinal-hippocampal resonances are predicted to dynamically stabilize the learning of entorhinal grid cells and hippocampal place cells, and parietal-prefrontal resonances are predicted to trigger the selective opening of basal ganglia gates to enable the read-out of context-appropriate actions. Grossberg (2013; 2016) reviews these and other aspects of ART as a cognitive and neural theory.

Recommended Reading

Amis GP, Carpenter GA (2010) Self-supervised ARTMAP. Neural Netw 23:265–282

Baloch AA, Grossberg S (1997) A neural model of high-level motion processing: line motion and formotion dynamics. Vis Res 37:3037–3059

Baloch AA, Grossberg S, Mingolla E, Nogueira CAM (1999) A neural model of first-order and second-order motion perception and magnocellular dynamics. J Opt Soc Am A 16:953–978

Berzhanskaya J, Grossberg S, Mingolla E (2007) Laminar cortical dynamics of visual form and motion interactions during coherent object motion perception. Spat Vis 20:337–395

Brown J, Bullock D, Grossberg S (1999) How the basal ganglia use parallel excitatory and inhibitory learning pathways to selectively respond to unexpected rewarding cues. J Neurosci 19:10502–10511

Brown JW, Bullock D, Grossberg S (2004) How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades. Neural Netw 17:471–510

Browning A, Grossberg S, Mingolla M (2009a) A neural model of how the brain computes heading from optic flow in realistic scenes. Cogn Psychol 59:320–356

Browning A, Grossberg S, Mingolla M (2009b) Cortical dynamics of navigation and steering in natural scenes: motion-based object segmentation, heading, and obstacle avoidance. Neural Netw 22:1383–1398

Bullock D, Grossberg S (1988) Neural dynamics of planned arm movements: emergent invariants and speed-accuracy properties during trajectory formation. Psychol Rev 95:49–90

Bullock D, Grossberg S (1991) Adaptive neural networks for control of movement trajectories invariant under speed and force rescaling. Hum Mov Sci 10:3–53

Bullock D, Cisek P, Grossberg S (1998) Cortical networks for control of voluntary arm movements under variable force conditions. Cereb Cortex 8:48–62

Cao Y, Grossberg S (2005) A laminar cortical model of stereopsis and 3D surface perception: closure and da Vinci stereopsis. Spat Vis 18:515–578

Cao Y, Grossberg S (2012) Stereopsis and 3D surface perception by spiking neurons in laminar cortical circuits: a method of converting neural rate models into spiking models. Neural Netw 26:75–98

Cao Y, Grossberg S, Markowitz J (2011) How does the brain rapidly learn and reorganize view- and positionally-invariant object representations in inferior temporal cortex? Neural Netw 24:1050–1061

Carpenter GA (1994) A distributed outstar network for spatial pattern learning. Neural Netw 7:159–168

Carpenter GA (1997) Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. Neural Netw 10:1473–1494

Carpenter GA (2001) Neural network models of learning and memory: leading questions and an emerging framework. Trends Cogn Sci 5:114–118

Carpenter GA, Gaddam SC (2010) Biased ART: a neural architecture that shifts attention toward previously disregarded features following an incorrect prediction. Neural Netw 23:435–451

Carpenter GA, Grossberg S (1987) A massively parallel architecture for a self-organizing neural pattern recognition machine. Comput Vis Graph Image Process 37:54–115

Carpenter GA, Grossberg S (1990) ART 3: hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. Neural Netw 4: 129–152

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

Carpenter GA, Markuzon N (1998) ARTMAP-IC and medical diagnosis: instance counting and inconsistent cases. Neural Netw 11:323–336

Carpenter GA, Grossberg S, Reynolds JH (1991a) ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. Neural Netw 4:565–588

Carpenter GA, Grossberg S, Rosen DB (1991b) Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system. Neural Netw 4:759–771

Carpenter GA, Grossberg S, Markuzon N, Reynolds JH, Rosen DB (1992) Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. IEEE Trans Neural Netw 3:698–713

Carpenter GA, Martens S, Ogas OJ (2005) Self-organizing information fusion and hierarchical knowledge discovery: a new framework using ARTMAP neural networks. Neural Netw 18:287–295

Chandler B, Grossberg S (2012) Joining distributed pattern processing and homeostatic plasticity in recurrent on-center off-surround shunting networks: noise, saturation, short-term memory, synaptic scaling, and BDNF. Neural Netw 25: 21–29

Chang H-C, Grossberg S, Cao Y (2014) Where's Waldo? How perceptual cognitive, and emotional brain processes cooperate during learning to categorize and find desired objects in a cluttered scene. Front Integr Neurosci doi:10.3389/fnint.2014.0043

Chapelle O, Schölkopf B, Zien A (eds) (2006) Semi-supervised learning. MIT, Cambridge

Contreras-Vidal JL, Grossberg S, Bullock D (1997) A neural model of cerebellar learning for arm movement control: cortico-spino-cerebellar dynamics. Learn Mem 3:475–502

Dranias M, Grossberg S, Bullock D (2008) Dopaminergic and non-dopaminergic value systems in conditioning and outcome-specific revaluation. Brain Res 1238:239–287

Elder D, Grossberg S, Mingolla M (2009) A neural model of visually guided steering, obstacle avoidance, and route selection. J Exp Psychol Hum Percept Perform 35:1501–1531

Fang L, Grossberg S (2009) From stereogram to surface: how the brain sees the world in depth. Spat Vis 22:45–82

Fazl A, Grossberg S, Mingolla E (2009) View-invariant object category learning, recognition, and search: how spatial and object attention are coordinated using surface-based attentional shrouds. Cogn Psychol 58:1–48

Foley NC, Grossberg S, Mingolla E (2012) Neural dynamics of object-based multifocal visual spatial attention and priming: object cueing, useful-field-of-view, and crowding. Cognitive Psychology 65: 77–117

Gancarz G, Grossberg G (1999) A neural model of the saccadic eye movement control explains task-specific adaptation. Vis Res 39:3123–3143

Grossberg S (1984) Some psychophysiological and pharmacological correlates of a developmental, cognitive, and motivational theory. In: Karrer R, Cohen J, Tueting P (eds) Brain and information: event related potential. New York Academy of Sciences, New York, pp 58–142.

Grossberg S (2000a) The complementary brain: unifying brain dynamics and modularity. Trends Cogn Sci 4:233–246

Grossberg S (2000b) The imbalanced brain: from normal behavior to schizophrenia. Biol Psychiatry 48:81–98

Grossberg S (2013) Adaptive resonance theory: how a brain learns to consciously attend, learn, and recognize a changing World. Neural Netw 37:1–47

Grossberg, S. (2016). Towards solving the hard problem of consciousness: the varieties of brain resonances and the conscious experiences that they support. Submitted for publication

Grossberg S, Huang T-R (2009) ARTSCENE: a neural system for natural scene classification. J Vis 9(6):1–19

Grossberg S, Merrill JWL (1992) A neural network model of adaptively timed reinforcement learning and hippocampal dynamics. Cogn Brain Res 1:3–38

Grossberg S, Merrill JWL (1996) The hippocampus and cerebellum in adaptively timed learning, recognition, and movement. J Cogn Neurosci 8:257–277

Grossberg S, Pearson L (2008) Laminar cortical dynamics of cognitive and motor working memory, sequence learning and performance: toward a unified theory of how the cerebral cortex works. Psychol Rev 115:677–732

Grossberg S, Pilly PK (2012) How entorhinal grid cells may learn multiple spatial scales from a dorsoventral gradient of cell response rates in a self-organizing map. PLoS Comput Biol 8(10):31002648. doi:10.1371/journal.pcbi.1002648

Grossberg S, Pilly PK (2014) Coordinated learning of grid cell and place cell spatial and temporal properties: multiple scales, attention, and oscillations. Philos Trans R Soc B 369:20120524

Grossberg S, Rudd ME (1992) Cortical dynamics of visual motion perception: short-range and long-range apparent motion (with Rudd ME). Psychol Rev 99:78–121

Grossberg S, Schmajuk NA (1989) Neural dynamics of adaptive timing and temporal discrimination during associative learning. Neural Netw 2:79–102

Grossberg S, Seidman D (2006) Neural dynamics of autistic behaviors: cognitive, emotional, and timing substrates. Psychol Rev 113:483–525

Grossberg S, Swaminathan G (2004) A laminar cortical model for 3D perception of slanted and curved surfaces and of 2D images: development, attention and bistability. Vis Res 44:1147–1187

Grossberg S, Vladusich T (2010) How do children learn to follow gaze, share joint attention, imitate their teachers, and use tools during social interactions? Neural Netw 23:940–965

Grossberg S, Leveille J, Versace M (2011) How do object reference frames and motion vector decomposition emerge in laminar cortical circuits? Atten Percept Psychophys 73:1147–1170

Grossberg S, Kuhlmann L, Mingolla E (2007) A neural model of 3D shape-from-texture: multiple-scale filtering, boundary grouping, and surface filling-in. Vis Res 47:634–672

Grossberg S, Mingolla E, Viswanathan L (2001) Neural dynamics of motion integration and segmentation within and across apertures. Vis Res 41:2521–2553

Grossberg S, Srihasam K, Bullock D (2012) Neural dynamics of saccadic and smooth pursuit eye movement coordination during visual tracking of unpredictably moving targets. Neural Netw 27:1–20

Huang T-R, Grossberg S (2010) Cortical dynamics of contextually cued attentive visual learning and search: spatial and object evidence accumulation. Psychol Rev 117:1080–1112

Hurvich LM, Jameson D (1957) An opponent-process theory of color vision. Psychol Rev 64: 384–390

Kelly FJ, Grossberg S (2000) Neural dynamics of 3-D surface perception: figure-ground separation and lightness perception. Percept Psychophys 62:1596–1619

Mhatre H, Gorchetchnikov A, Grossberg S (2012) Grid cell hexagonal patterns formed by fast self-organized learning within entorhinal cortex. Hippocampus 22:320–334

Pack C, Grossberg S, Mingolla E (2001) A neural model of smooth pursuit control and motion perception by cortical area MST. J Cogn Neurosci 13:102–120

Pilly PK, Grossberg S (2012) How do spatial learning and memory occur in the brain? Coordinated learning of entorhinal grid cells and hippocampal place cells. J Cogn Neurosci 24:1031–1054

Pilly PK, Grossberg S (2014) How does the modular organization of entorhinal grid cells develop? Front Hum Neurosci. doi:10.3389/fnhum.2014.0037

Schiller PH (1982) Central connections of the retinal ON and OFF pathways. Nature 297:580–583

Simone G, Farina A, Morabito FC, Serpico SB, Bruzzone L (2002) Image fusion techniques for remote sensing applications. Inf Fusion 3:3–15

Srihasam K, Bullock D, Grossberg S (2009) Target selection by frontal cortex during coordinated saccadic and smooth pursuit eye movements. J Cogn Neurosci 21:1611–1627

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 artificial, 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
Faculty of Information Technology, 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 machine learning and data mining 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 in general across 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, ▶ holdout evaluation, ▶ out-of-sample evaluation and ▶ prospective evaluation.

Cross-References

▶ Evaluation of Learning Algorithms
▶ Model Evaluation

References

Hastie T, Tibshirani R, Friedman JH (2001) The elements of statistical learning. Springer, New York
Mitchell TM (1997) Machine learning. McGraw-Hill, New York
Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading
Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196
Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco

Analogical Reasoning

▶ Instance-Based Learning

Analysis of Text

▶ Text Mining

Analytical Learning

▶ Deductive Learning
▶ Explanation-Based Learning

Anomaly Detection

Varun Chandola[1], Arindam Banerjee[2], and Vipin Kumar[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]University of Minnesota, Minneapolis, MN, USA

Abstract

Anomalies correspond to the behavior of a system which does not conform to its expected or normal behavior. Identifying such anomalies from observed data, or the task of anomaly detection, is an important and often critical analysis task. This includes finding abnormalities in a medical image, fraudulent transactions in a credit card history, or structural defects in an aircraft's engine. The importance of this problem has resulted in a large body of literature on this topic. However, given that the definition of an anomaly is strongly tied to the underlying application, the existing research is often embedded in the application domains, and it is unclear how methods developed for one domain would perform in another. The goal of this article is to provide a general introduction of the anomaly detection problem. We start with the basic formulation of the problem and then discuss the various extensions. In particular, we discuss the challenges associated with identifying anomalies in structured data and provide an overview of existing research in this area. We hope that this article will provide a better understanding of the different directions in which research has been done on this topic, and how techniques developed in one area can be applied in domains for which they were not intended to begin with.

Introduction

Anomalies are the unusual, unexpected, surprising patterns in the observed world. Identifying, understanding, and predicting anomalies from data form one of the key pillars of modern data mining. Effective detection of anomalies allows extracting critical information from data which can then be used for a variety of applications, such as to stop malicious intruders, detect and repair faults in complex systems, and better understand the behavior of natural, social, and engineered systems.

Anomaly detection refers to the problem of finding anomalies in data. While "anomaly" is a generally accepted term, other synonyms, such as outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants, are often used in different application domains. In particular, anomalies and outliers are often used interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance or healthcare, intrusion detection for cybersecurity, fault detection in safety critical systems, and military surveillance for enemy activities. The importance of anomaly detection stems from the fact that for a variety of application domains, anomalies in data often translate to significant (and often critical) actionable insights. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination (Kumar 2005). An anomalous remotely sensed weather variable such as temperature could imply a heat wave or cold snap or even faulty remote sensing equipment. An anomalous MRI image may indicate early signs of Alzheimer's or the presence of malignant tumors (Spence et al. 2001). Anomalies in credit card transaction data could indicate credit card or identity theft (Aleskerov et al. 1997), or anomalous readings from a spacecraft sensor could signify a fault in some component of the spacecraft (Fujimaki et al. 2005).

Anomaly detection is generally considered as a core machine learning or data mining problem, in the same vein as classification and clustering. Given the practical significance of anomalies, there has been a tremendous interest in studying this problem, starting from statistical methods proposed as early as the nineteenth century (Edgeworth 1887). Over time, a variety of anomaly detection techniques have been developed in several research communities. Many of these techniques have been specifically developed for certain application domains, while others are more generic. Several books and surveys have been published in recent years that provide an overview of the vast literature on this topic (Chandola et al. 2009; Aggarwal 2013; Hodge and Austin 2004; Chandola et al. 2012; Akoglu et al. 2015).

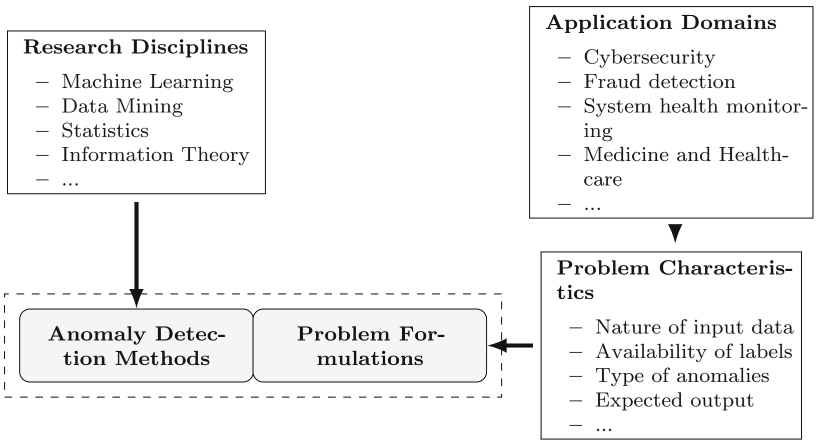
However, one key characteristic of anomaly detection sets it apart from other machine learning problems. Anomaly detection is a highly application-oriented problem which means that there is a lack of a consistent definition of an anomaly across tasks and application domains. Researchers typically define an anomaly in a way that best suits the target application. Thus, several different formulations of the anomaly detection problem exist. Existing solutions for these problem formulations have borrowed concepts from a variety of disciplines in mathematics, statistics, and computer science. This has resulted in a rich and complex landscape for anomaly detection research (See Fig. 1).

The goal of this article is to provide the readers a general understanding of the complex problem space of anomaly detection. Starting with the most basic problem setting, i.e., identifying anomalous data instances from a data set, we then discuss other formulations and the corresponding methods. To highlight the practical importance of anomaly detection, we provide an application-oriented overview of existing methods. Finally, we discuss open challenges and research questions that exist in this area to motivate future research.

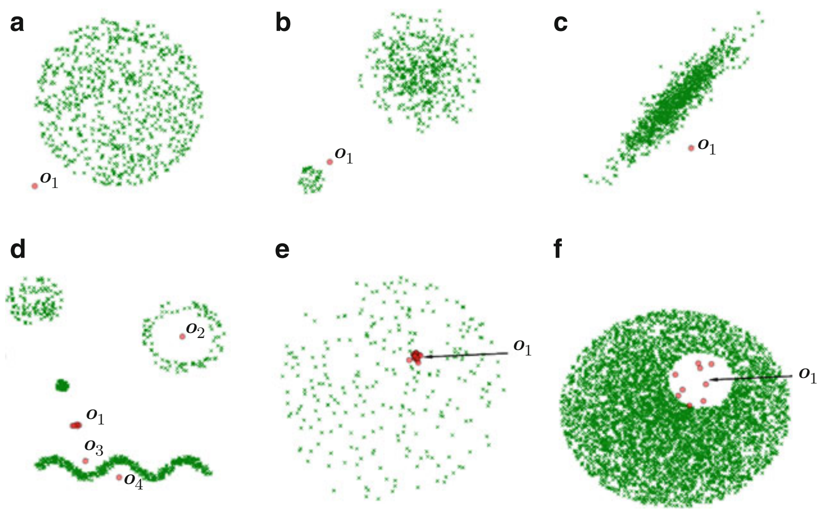
Point Anomaly Detection

In the most widely accepted setting for anomaly detection, also referred to as *point anomaly detection*, the goal is to identify points (objects, instances, etc.) in a data set that do not conform to the accepted normal behavior. Typically, no other knowledge about the normal or anomalous behavior is available. The lack of any ground truth for training makes this an *unsupervised anomaly detection problem*. In the sequel, we briefly talk about other formulations in which partial knowledge of normal and/or anomalous behavior is available.

Even in the basic setting of point anomaly detection, a uniform definition of anomaly does not exist. Figure 2 shows several hypothetical examples of anomalies in a two-dimensional data set. In each of the example, anomalies have a different interpretation. For instance, the point o_1 in Fig. 2a is anomalous because it is far away from the rest of the data points which belong to a dense region. In Fig. 2b, however, the point o_1



Anomaly Detection, Fig. 1 Anatomy of an anomaly detection problem



Anomaly Detection, Fig. 2 Examples of anomalies in 2-D data. (**a**) Anomaly with respect to rest of the data points. (**b**) Anomaly with respect to local neighborhood. (**c**) Anomaly with respect to the data distribution. (**d**) Anomaly with respect to local dense regions. (**e**) Anomalous tight cluster in a sparse region. (**f**) Anomalous sparse cluster in a dense region

is anomalous because it lies relatively far away from a dense region, even though there are points in the second sparse region which are equally distant from their nearest points. In the third example (see Fig. 2c), the point o_1 is anomalous because it lies away from the statistical distribution (bivariate normal) of the data. On the other hand, there are several points located at the ends of the elliptical distribution that are farther away from the points that o_1. The anomalies o_2, o_3, and o_4 in Fig. 2d are points that are away from their closest dense regions. The points in the anomalous set o_1 in Fig. 2d, e, and f are all groups of points whose density is anomalous with respect to the rest of the data set. As shown in the above simple 2-D example, even for point anomaly detection, one can define anomalies in multiple ways. Most existing anomaly detection methods, on the other hand, have been developed, by starting from a different notion of anomaly often motivated by a specific application domain. Thus, one method might be successful in one scenario and not in the other. We now discuss some prominent classes of point anomaly detection methods and the definitions of anomalies that they are best suited for. The various classes of point anomaly detection methods are briefly discussed below:

Nearest neighbor-based methods analyze the nearest neighborhood of a test instance to assign it an anomaly score (Ramaswamy et al. 2000; Knorr and Ng 1999; Knorr et al. 2000; Otey et al. 2006; Tang et al. 2002; Breunig et al. 2000, 1999). The key assumption underlying nearest neighbor-based anomaly detection methods is that normal points lie in dense neighborhoods and anomalous points lie in sparse neighborhoods. Nearest neighbor methods consider suitable measures of density, e.g., distance to the k-th nearest neighbor (Ramaswamy et al. 2000), radius needed to enclose a certain number of points (Knorr and Ng 1999; Knorr et al. 2000), etc. Such methods are capable of identifying global anomalies (See Fig. 2a) but are shown to perform poorly when the data has regions with varying densities (See Fig. 2b). For such scenarios, methods such as local outlier factor (Breunig et al. 1999) and commute distance-based outlier factor (Khoa and Chawla 2010) have been proposed. When data is high dimensional, such methods typically suffer from the "curse of dimensionality." Methods such as angle-based outlier detection (Kriegel et al. 2008) and subspace-based approaches (Zhang and Wang 2006) have been proposed to address this issue.

Clustering-based methods learn clusters from a given data set and assign an anomaly score to a test instance based on its relationship with its nearest cluster (Eskin et al. 2002; He et al. 2003; Marchette 1999; Eskin et al. 2002; Portnoy et al. 2001; Mahoney et al. 2003). Clustering-based methods assume that while normal points exhibit cluster structure, anomalous points do not belong to a cluster or are far away from the nearest normal cluster representative. In certain settings, if the anomalies themselves may form a cluster, one assumes that normal points form large and dense clusters, whereas anomalous points form small clusters or clusters with low density (see Fig. 2d, e and f). While such methods identify anomalies as a post-clustering phase, recently, there have been methods that focus on identifying anomalies simultaneously with the clusters (Ott et al. 2014; Chawla and Gionis 2013).

Statistical methods estimate a parametric or nonparametric model from the data and apply a statistical test on the probability of the instance to be generated by the estimated model to assign an anomaly score to the test instance (Barnett and Lewis 1994; Fox 1972; Abraham and Chuang 1989; Laurikkala et al. 2000; Chow and Yeung 2002). Such statistical models assume that normal points appear in the high probability regions of the distribution, thereby having high likelihood of occurring and hence low anomaly scores. On the other hand, anomalous points appear in the low probability regions of the distribution and have high anomaly score. Such methods are effective if the normal instances can be modeled by a statistical distribution. For instance, if the data in Fig. 2c is modeled as a bivariate normal distribution, the anomalous point o_1 can be easily identified using a standard Mahalanobis statistic, while rest of the points will appear normal.

Classification-based methods learn a classifier from a labeled (or unlabeled) training data and assign an anomaly score or label to a test data instance (Tax 2001; Tax and Duin 1999a, b; Barbara et al. 2001; Roth 2004; Hawkins et al. 2002; Mahoney and Chan 2002, 2003). The key assumption underlying classification-based anomaly detection methods is that based on the available training data, one can learn a classifier in the given feature space to distinguish between normal and anomalous points. Classification-based anomaly detection methods can be categorized into one-class methods, which have one model for the normal class and any point which does not fit that model is deemed anomalous, and multi-class methods, which have multiple normal classes and points which do not fit any of the normal classes are deemed anomalous. A variety of models such as support vector machines, neural networks, Bayesian models, and rule-based systems have been used for classification-based anomaly detection. However, such methods are limited by their dependence on availability of labels for normal and/or anomalous behavior. There are, however, methods that can operate in a purely unsupervised setting, such as the one-class support vector machines (Schölkopf et al. 2001; Tax 2001).

Spectral decomposition-based methods find an approximation of the data using a combination of attributes that capture the bulk of variability in the data. Instances that are significantly different from others in the lower approximation are detected as anomalies (Agovic et al. 2007; Parra et al. 1996; Shyu et al. 2003; Fujimaki et al. 2005). Such methods are particularly effective in scenarios where the data is being generated from a lower dimensional manifold, e.g., See Fig. 2c.

Information theoretic methods are based on the assumption that anomalies in data induce irregularities in the information content of the data set. Such methods analyze the *information content* of a data set using different information theoretic measures such as *Kolmogorov complexity, entropy, relative entropy*, etc. and detect instance that induces irregularities in the information content of the data set as anomalies (Arning et al. 1996; Keogh et al. 2004; Lee and Xiang 2001; He et al. 2005, 2006).

Extensions to Point Anomaly Detection

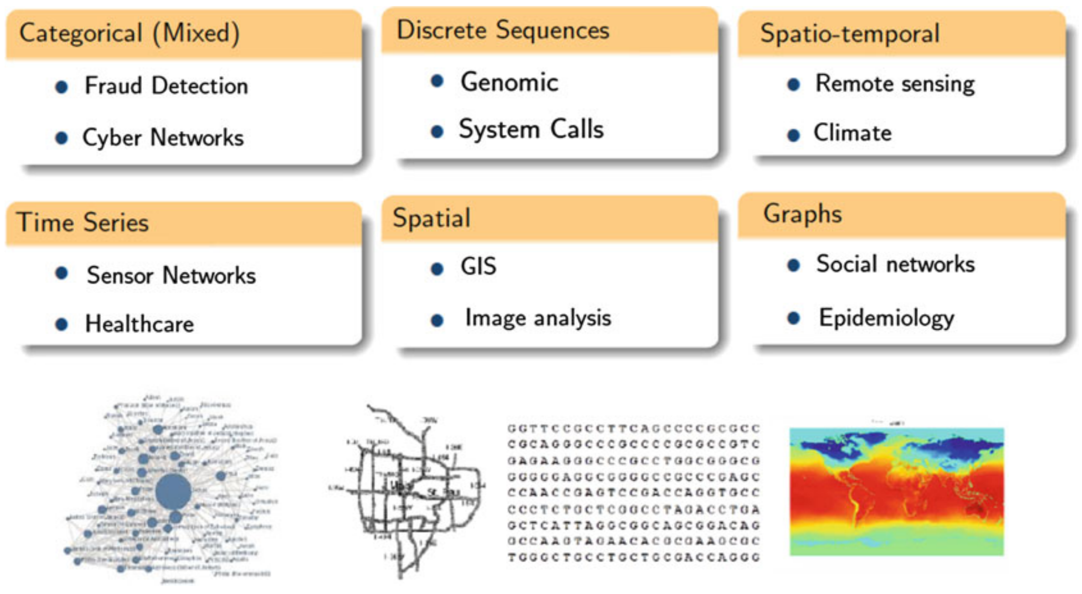
In certain settings, the unsupervised point anomaly detection problem discussed in section "Point Anomaly Detection" is not rich enough to capture all requirements of an application domain. Here we discuss some of the different ways in which the basic problem setting is typically extended.

Nature of Input Data

The modality of the data determines the applicability of anomaly detection techniques. For example, for statistical techniques, different statistical models have to be used for continuous and categorical data. Similarly, for nearest neighbor-based techniques, the nature of attributes would determine the distance measure to be used. Often, instead of the actual data, the pairwise distance between instances might be provided in the form of a distance (or similarity) matrix. In such cases, techniques that require original data instances are not applicable, e.g., many statistical methods and certain classification-based techniques. However, many of the nearest neighbor-based or clustering-based methods discussed in section "Point Anomaly Detection" are still applicable.

Input data can also be categorized based on the relationship present among data instances (Tan et al. 2005). Most of the existing anomaly detection techniques deal with data represented as a vector of attributes (record or point data, if the data can be mapped onto a coordinate space), as discussed in section "Point Anomaly Detection." Typically, no relationship is assumed among the data instances.

In general, data instances can be related to each other. Some examples are *sequence data*, *spatial data*, and *graph data* (See Fig. 3 for an overview). In sequence data, the data instances are linearly ordered, e.g., time-series data, genome sequences, protein sequences. In *spatial data*, each data instance is related to its neighboring instances, e.g., vehicular traffic data, ecological data. When the spatial data has a temporal (sequential) component, it is referred to as *spatiotemporal* data, e.g., climate data. In *graph data*, data instances are represented as vertices in a graph and are connected to other vertices with edges. Later in this section, we will



Anomaly Detection, Fig. 3 Complex data types encountered by anomaly detection and some sample application domains discuss situations where such relationship among data instances becomes relevant for anomaly detection.

Type of Anomaly

Anomaly detection techniques vary depending on the nature of the desired anomaly. We have already discussed point anomalies in section "Point Anomaly Detection," which is the most common form of anomaly. While point anomalies are isolated by nature, several applications need to consider anomalies in a context or small collection of observations which appear anomalous. One can define two additional types of anomalies to capture such structures: contextual anomalies and collective anomalies.

Contextual Anomalies

Data instances which are anomalous in a specific context, but not otherwise, are called contextual anomaly (also referred to as *conditional anomaly* Song et al. 2007). For example, a temperature of 70 °F may be normal over summer, but is anomalous in the context of winter; a heart rate of 130 may be normal for an individual exercising or running, but is anomalous when the individual is resting. In the setting of contextual anomaly detection, the context, such as summer/winter and exercising/resting, has to be specified as a part of the problem formulation. In particular, the data instances are defined using following two sets of attributes:

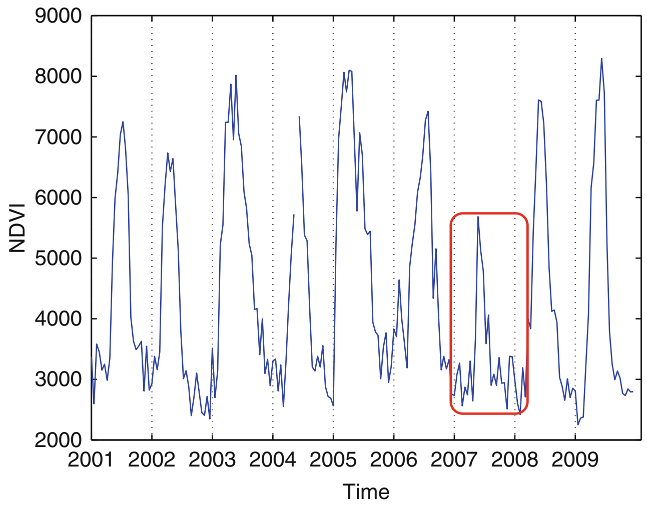
1. *Contextual attributes*. The contextual attributes are used to determine the context (or neighborhood) for that instance. For example, in spatial data sets, the longitude and latitude of a location are the contextual attributes. In time-series data, time is a contextual attribute which determines the position of an instance on the entire sequence.
2. *Behavioral attributes*. The behavioral attributes define the non-contextual characteristics of an instance. For example, in a spatial data set describing the average rainfall of the entire world, the amount of rainfall at any location is a behavioral attribute.

The context determines the normal behavioral attributes, and the normal can be different in different contexts. Anomalous behavior is determined using the values for the behavioral attributes within a specific context, in particular when such values deviate from what is normal in that context. A data instance might be a contextual anomaly in a given context, but an identical data instance (in terms of behavioral attributes) could be considered normal in a different context. This property is key in identifying contextual and behavioral attributes for a contextual anomaly detection technique.

Contextual anomalies have been most commonly explored in time-series data (Weigend et al. 1995; Salvador and Chan 2003) and spatial data (Kou et al. 2006; Shekhar et al. 2001). In spatial data domain, an observation has a neighborhood specified by its location component (refer to our earlier discussion on spatial data). Consider an example in which each data instance is a county location which is defined over several attributes. If these attributes show high pollution levels for a particular county, but the neighborhood of this county is also highly polluted, then this county is not an anomaly. But if the neighborhood has very low pollution, then this county becomes an anomaly.

A similar example can be found in the credit card fraud detection domain. A contextual attribute in credit card domain can be the *time* of purchase. Suppose an individual usually has a weekly shopping bill of $100 except during the Christmas week, when it reaches $1000. A new purchase of $1000 in a week in July will be considered a contextual anomaly, since it does not conform to the normal behavior of the individual in the context of time (even though the same amount spent during Christmas week will be considered normal).

The choice of applying a contextual anomaly detection technique is determined by the meaningfulness of the contextual anomalies in the target application domain. Another key factor is the availability of *contextual* attributes. In several cases, defining a context is straightforward, and hence applying a contextual anomaly detection technique makes sense. In other cases, defining



Anomaly Detection, Fig. 4 Example of a collective anomaly—MODIS NDVI Time Series for 2001–2009 for a Southern California location with a known forest fire (*Canyon fire*) in 2007 [*src:* http://cdfdata.fire.ca.gov/incidents/incidents_archived?archive_year=2007]

a context is not easy, making it difficult to apply such techniques.

Collective Anomalies

If a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous. Figure 4 illustrates an example which shows a greenness measurement called normalized difference vegetation index (NDVI) for a geographic location obtained from a satellite instrument (MODIS). The highlighted region denotes an anomaly where the greenness values are abnormally low for the entire year of 2007 due to a wildfire during that time. Note that the individual measurements during the year are not anomalous by themselves.

As an another illustrative example, consider a sequence of actions occurring in a computer as shown below:

... http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, **ssh**, **buffer-overflow**, **ftp**, http-web, ftp, smtp-mail,http-web ...

The highlighted sequence of events (**buffer-overflow**, **ssh**, **ftp**) corresponds to a typical web-based attack by a remote machine followed by copying of data from the host computer to remote destination via *ftp*. It should be noted that this collection of events is an anomaly, but the individual events are not anomalies when they occur in other locations in the sequence.

Collective anomalies have been explored for sequence (discrete and time series) data (Forrest et al. 1999; Sun et al. 2006), graph data (Noble and Cook 2003; Li et al. 2014; Akoglu et al. 2015), and spatial data (Shekhar et al. 2001). It should be noted that while point anomalies can occur in any data set, collective anomalies can occur only in data sets in which data instances are related. In contrast, occurrence of contextual anomalies depends on the availability of context attributes in the data. A point anomaly or a collective anomaly can also be a contextual anomaly if analyzed with respect to a context. Thus a point anomaly detection problem or collective anomaly detection problem can be transformed to a contextual anomaly detection problem by incorporating the context information.

Data Labels

In some scenarios, labels associated with data instances denote if that instance is *normal* or *anomalous*. (Also referred to as normal and anomalous classes.) Obtaining labeled data which is accurate as well as representative of all types of normal and anomalous behaviors is often prohibitively expensive. Labels are often provided by human domain experts and hence usually require substantial effort and time. Even in settings where a human expert is able to provide labels, it is usually easier to give examples of normal instances, since the number of different ways an anomaly can occur is quite large and finding examples of the different types of anomalies is difficult. Further, anomalous behavior is often dynamic in nature, e.g., new types of anomalies might arise, for which there is no labeled training data. In certain cases, such as aviation safety, anomalous instances would translate to catastrophic events, and hence will be very rare.

Based on the extent to which the labels are available, anomaly detection techniques can operate in one of the following three modes: supervised, semi-supervised, and unsupervised anomaly detection. Several of the anomaly detection methods discussed in section "Point Anomaly Detection" are unsupervised methods. Semi-supervised methods typically assume availability of a training data that represents the normal behavior. The general approach for such methods is to construct a statistical or machine learning model of normal behavior and then apply a statistical or proximity test to detect new instances which are not consistent with the learnt model. Supervised methods assume availability training data that represents both normal and anomalous behavior. Typically, anomalous events have a much smaller prior probability, and one can leverage methods for rare-class classification, cost-sensitive classification, and other ways of handling class imbalance. However, such methods find limited applicability since obtaining representative training data for anomalous behavior is typically infeasible.

Output of Algorithm

An important aspect for any anomaly detection technique is the manner in which the anomalies are reported. Typically, the outputs produced by anomaly detection techniques are one of the following two types: scores or binary predictions. Scores allow analysts to rank the anomalies in terms of the severity. Typically a threshold is then applied to the scores to identify the anomalies on which to act upon. The threshold is often set by either identifying a natural cutoff point in the sorted scores or based on the number of desired anomalies for further analysis. Methods that assign a binary label to data objects (anomaly or normal) are often easier to understand, though they lack the capability of ranking the anomalies. There are some research in calibrating the scores as probabilities (Gao and Tan 2006) for better interpretability (Kriegel et al. 2011; Schubert et al. 2012).

Anomaly Detection for Complex Data

In section "Point Anomaly Detection" we discussed anomaly detection in the context of data without any explicit relationship defined among them. However, in many applications, data objects are related, and often the anomalous behavior can only be identified by analyzing the relationships between the objects. In this section, we discuss the anomaly detection methods developed to handle the different types of relationships.

Symbolic Sequences

In this section, we provide an overview of the existing research on anomaly detection for symbolic sequences. Methods in this area can be grouped into following categories:

– **Kernel-Based Techniques:** These techniques treat the entire test sequence as a unit element in the analysis (Budalakoti et al. 2006, 2007; Yang and Wang 2003) and hence are analogous to point-based anomaly detection techniques. They typically apply a proximity-based point anomaly detection technique by defining an appropriate similarity kernel for the sequences.
– **Window-Based Techniques:** These techniques analyze a short window of symbols—a short subsequence—within the test sequence at a time (Forrest et al. 1996; Hofmeyr et al. 1998; Endler 1998; Debar et al. 1998; Ghosh et al. 1999a, b; Lane and Brodley 1997, 1999; Cabrera et al. 2001). Thus such techniques treat a subsequence within the test sequence as a unit element for analysis. These techniques require an additional step

in which the anomalous nature of the entire test sequence is determined, based on the analysis on the subsequences within the entire sequence.

- **Markovian Techniques:** These techniques predict the probability of observing each symbol of the test sequence, using a probabilistic model, and use the per-symbol probabilities to obtain an anomaly score for the test sequence (Sun et al. 2006; Ye 2004; Michael and Ghosh 2000; Eskin et al. 2001; Lee et al. 1997). These techniques analyze each symbol with respect to previous few symbols.
- **Hidden Markov Model-Based Techniques:** These techniques transform the input sequences into sequences of hidden states and then detect anomalies in the transformed sequences (Forrest et al. 1999; Qiao et al. 2002; Zhang et al. 2003; Florez-Larrahondo et al. 2005).

Though several techniques have been proposed for symbolic sequences in various application domains, there has not been any cross domain evaluation and understanding of the existing techniques. Forrest et al. (1999) compared four different anomaly detection techniques, but evaluated them in the context of system call intrusion detection. Sun et al. (2006) proposed a technique for protein sequences, but no evaluation with techniques proposed for system call data was done. Similarly, while Budalakoti et al. (2006) proposed a clustering-based techniques to detect anomalies in flight sequences, it has not been shown how the same technique would perform on system call intrusion detection data or protein data.

Most of the above methods identify an anomalous sequence from a set of sequences, assuming that majority of the sequences are normal. Other methods focus on a different problem formulation, also referred to as *discord detection*, in which the goal is to identify a subsequence within a long sequence which is anomalous with respect to the rest of the sequence. Most of the existing techniques that handle this problem formulation slide a fixed length window across the given long sequence and compare each window with the remaining sequence to detect anomalous windows (Keogh et al. 2005a, 2006; Lin et al. 2005; Wei et al. 2005).

Time Series

Most methods that handle time-series data deal primarily with univariate signals, i.e., a single measurement captured over time. Several statistical techniques detect anomalous observations (also referred to as *outliers*) within a single time series using various time series modeling techniques such as regression (Fox 1972; Abraham and Chuang 1989; Rousseeuw and Leroy 1987), autoregression (AR) (Fujimaki et al. 2005; Wu and Shao 2005), ARMA (Pincombe 2005), ARIMA (Zare Moayedi and Masnadi-Shirazi 2008), support vector regression (SVR) (Ma and Perkins 2003), Kalman filters Knorn and Leith (2008), etc. The general approach behind such techniques is to forecast the next observation in the time series, using the statistical model and the time series observed so far, and compare the forecasted observation with the actual observation to determine if an anomaly has occurred.

Two broad categories of techniques have been proposed to identify anomalous time series in a time-series database (Chandola et al. 2009), viz., segmentation-based and kernel-based anomaly detection techniques. The general approach behind segmentation-based techniques is to segment the normal time series and treat each segment as a state in a *finite-state automaton* (FSA) and then use the FSA to determine if a test time series is anomalous or not. Several variants of the segmentation-based technique have been proposed (Chan and Mahoney 2005; Mahoney and Chan 2005; Salvador and Chan 2005). Kernel-based anomaly detection techniques compute similarity/distance between time series and apply a nearest neighbor-based anomaly detection technique on the similarity "kernel" (Protopapas et al. 2006; Wei et al. 2006; Yankov et al. 2007). Protopapas et al. (2006) use *cross correlation* as the similarity measure and compute the anomaly score of a test time series as the inverse of its average similarity to all other time series in the given data set. Wei et al. (2006) use a *rotation invariant* version of Euclidean distance to compute distance between time series and then assign an anomaly score to each time series as equal to its distance to its nearest neighbor. Yankov et al. (2007) proposed pruning-based heuristics to improve the efficiency of the nearest neighbor technique (Wei et al. 2006).

Several anomaly detection techniques for time series data identify anomalous subsequences within a long time series (also referred to as *discords*) (Keogh et al. 2004, 2005a, 2006; Lin et al. 2005; Fu et al. 2006; Bu et al. 2007; Yankov et al. 2007). Such techniques analyze fixed length windows obtained from the time series by comparing each window with the rest of the time series or against all other windows from that time series. A window which is significantly different from other windows is declared as a discord.

Limited research has been done to identify anomalies in multivariate time series data. Most existing methods for multivariate time series focus on detecting a single anomalous multivariate observation (Baragona and Battaglia 2007; Galeano et al. 2004; Tsay et al. 2000). Baragona and Battaglia (2007) propose an ICA-based technique to detect outliers in multivariate time series. The underlying idea is to isolate the multivariate time series into a set of independent univariate components and an outlier signal and analyze the univariate outlier signal to determine the outliers. The ICA-based technique assumes that the observed signals are linear combination of independent components as well as independent noise signal, and the added noise has a high kurtosis.

Cheng et al. (2009) proposed a distance-based approach to detect anomalous subsequences within a given multivariate sequence. For a given multivariate sequence S, all w length windows are extracted. The distance between each pair of windows is computed to obtain a symmetric $(T - w + 1) \times (T - w + 1)$ kernel matrix. A fully connected graph is constructed using the kernel matrix in which each node represents a w length window and the weight on the edges between the pair of windows is equal to the similarity (inverse of distance) between the pair. The nodes (or components) of the graph that have least connectivity are declared as anomalies.

Graphs and Networks

There has been considerable work done in the area of anomaly detection in graphs (Akoglu et al. 2015). Two broad categories of methods exist for detecting anomalies in graphs. The first type of methods looks for anomalous substructures or patterns within a graph (collective anomalies), while the second type of methods focuses on identifying anomalous nodes (contextual anomalies).

The first type of methods typically operates on graphs in which the nodes and/or edges are described using a set of attributes. Such graphs are often referred to as attributed graphs. The general approach here is to identify subgraphs within a large graph that have similar distribution of attributes (Noble and Cook 2003; Eberle and Holder 2007). In particular, the work by Noble and Cook (2003) identifies the frequent subgraphs in a graph with categorical attributes. Any subgraph that does not match the frequent subgraphs is considered to be anomaly. Subsequently, several variants of the original method have been proposed (Gao et al. 2010; Li et al. 2014; Sánchez et al. 2014).

The second type of methods analyzes each node in a graph with respect to its neighborhood. For instance, the OddBall method (Akoglu et al. 2010), analyzes each node with respect to its *egonet* which is the subgraph induced by the node and the other nodes connected to it in the graph. Other similar methods identify nodes that do not belong to densely connected communities (Sun et al. 2005; Ding et al. 2012; Tong and Lin 2011). Similar methods have been proposed to identify anomalies in attributed graphs (Gao et al. 2010; Müller et al. 2013).

Conclusions and Future Directions

The notion of anomaly is important in most real world settings. Data-driven methods for timely and effective identification of anomalies are essential, and this has triggered tremendous interest in the research community. However, the key difference between anomaly detection and other machine learning problems such as classification and clustering is the lack of a consistent definition of anomalies. Instead, several definitions of anomalies exist, each tailored to the need of an underlying application. In this article, we have provided an overview of this rich area by discussing the key aspects of this problem.

We have discussed different classifications of anomaly detection methods and provided understanding of the strengths and weaknesses of these classes of methods. One of the important subareas in this context is the class of methods that handle complex structured data. We have discussed methods that have been specifically developed to handle sequences, time series, and network data.

Given the unstructured nature of current research, a theoretical understanding of the anomaly detection is challenging to obtain. A possible future work would be to unify the assumptions made by different techniques regarding the normal and anomalous behavior into a statistical or machine learning framework. A limited attempt in this direction is provided by Knorr et al. (1997), where the authors show the relation between distance based and statistical anomalies for two-dimensional data sets.

There are several promising directions for further research in anomaly detection. Contextual and collective anomaly detection techniques are beginning to find increasing applicability in several domains, and there is much scope for development of new techniques in this area. The presence of data across different distributed locations has motivated the need for distributed anomaly detection techniques (Zimmermann and Mohay 2006). While such techniques process information available at multiple sites, they often have to simultaneously protect the information present in each site, thereby requiring privacy preserving anomaly detection techniques (Vaidya and Clifton 2004). With the emergence of sensor networks, processing data as it arrives has become a necessity. Many techniques discussed in this article require the entire test data before detecting anomalies. Recently, techniques have been proposed that can operate in an online fashion (Pokrajac et al. 2007); such techniques not only assign an anomaly score to a test instance as it arrives, but also incrementally update the model.

References

Abraham B, Chuang A (1989) Outlier detection and time series modeling. Technometrics 31(2):241

Aggarwal CC (2013) Outlier analysis, Springer, New York

Agovic A, Banerjee A, Ganguly AR, Protopopescu V (2007) Anomaly detection in transportation corridors using manifold embedding. In: First international workshop on knowledge discovery from sensor data, ACM Press, New York

Akoglu L, McGlohon M, Faloutsos C (2010) OddBall: spotting anomalies in weighted graphs. In: In Pacific-Asia conference on knowledge discovery and data mining (PAKDD), Hyderabad

Akoglu L, Tong H, Koutra D (2015) Graph based anomaly detection and description: a survey. Data Min Knowl Discov 29(3):626

Aleskerov E, Freisleben B, Rao B (1997) Cardwatch: a neural network based database mining system for credit card fraud detection. In: Proceedings of IEEE computational intelligence for financial engineering, New York, pp 220–226

Arning A, Agrawal R, Raghavan P (1996) A linear method for deviation detection in large databases. In: Proceedings of 2nd international conference of knowledge discovery and data mining, pp 164–169. citeseer.ist.psu.edu/arning96linear.html

Baragona R, Battaglia F (2007) Outliers detection in multivariate time series by independent component analysis. Neural Comput 19(7):1962. doi:http://dx.doi.org/10.1162/neco.2007.19.7.1962

Barbara D, Couto J, Jajodia S, Wu N (2001) Detecting novel network intrusions using bayes estimators. In: Proceedings of the first SIAM international conference on data mining, Chicago

Barnett V, Lewis T (1994) Outliers in statistical data, Wiley, Chichester

Breunig MM, Kriegel HP, Ng RT, Sander J (1999) Optics-of: identifying local outliers. In: Proceedings of the third European conference on principles of data mining and knowledge discovery, Springer, Berlin/New York, pp 262–270

Breunig MM, Kriegel HP, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of 2000 ACM SIGMOD international conference on management of data. ACM Press, pp 93–104. doi:http://doi.acm.org/10.1145/342009.335388

Bu Y, Leung TW, Fu A, Keogh E, Pei J, Meshkin S (2007) WAT: finding top-k discords in time series database. In: Proceedings of 7th siam international conference on data mining

Budalakoti S, Srivastava A, Akella R, Turkov E (2006) Anomaly detection in large sets of high-dimensional symbol sequences. Technical report NASA TM-2006-214553, NASA Ames Research Center

Budalakoti S, Srivastava A, Otey M (2007) Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. In: Proceedings of the IEEE international conference on systems, man, and cybernetics, Montreal, vol. 37

Cabrera JBD, Lewis L, Mehra RK (2001) Detection and classification of intrusions and faults using sequences of system calls. SIGMOD Records 30(4):25. doi:http://doi.acm.org/10.1145/604264.604269

Chan PK, Mahoney MV (2005) Modeling multiple time series for anomaly detection. In: Proceedings of the fifth IEEE international conference on data mining. IEEE Computer Society, Washington, DC, pp 90–97

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection a survey. ACM Comput Surv 41(3):15:1–15:58

Chandola V, Banerjee A, Kumar V (2012) Anomaly detection for discrete sequences: a survey. IEEE Trans Knowl Data Eng 24:823. doi:http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.235

Chandola V, Cheboli D, Kumar V (2009) Detecting anomalies in a timeseries database. Technical report 09-004, Computer Science Department, University of Minnesota

Chawla S, Gionis A (2013) k-means-: a unified approach to clustering and outlier detection. In: Proceedings of the 13th SIAM international conference on data mining, Austin, 2–4 May 2013, pp 189–197

Cheng H, Tan PN, Potter C, Klooster S (2009) Detection and characterization of anomalies in multivariate time series. In: Proceedings of the ninth SIAM international conference on data mining (SDM)

Chow C, Yeung DY (2002) Parzen-window network intrusion detectors. In: Proceedings of the 16th International conference on pattern recognition, vol 4. IEEE Computer Society, Washington, DC, p 40385

Debar H, Dacier M, Nassehi M, Wespi A (1998) Fixed vs. variable-length patterns for detecting suspicious process behavior. In: Proceedings of the 5th European symposium on research in computer security, Springer, London, pp 1–15

Ding Q, Katenka N, Barford P, Kolaczyk E, Crovella M (2012) Intrusion as (anti)social communication: characterization and detection. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'12), pp 886–894

Eberle W, Holder L (2007) Anomaly detection in data represented as graphs. Intell Data Anal 11(6):663. http://dl.acm.org/citation.cfm?id=1368018.1368024

Edgeworth FY (1887) On discordant observations. Philos Mag 23(5):364

Endler D (1998) Intrusion detection: applying machine learning to solaris audit data. In: Proceedings of the 14th annual computer security applications conference. IEEE Computer Society, Los Alamitos, p 268

Eskin E, Arnold A, Prerau M, Portnoy L, Stolfo S (2002) A geometric framework for unsupervised anomaly detection. In: Proceedings of applications of data mining in computer security. Kluwer Academics, Dordrecht, pp 78–100

Eskin E, Lee W, Stolfo S (2001) Modeling system call for intrusion detection using dynamic window sizes. In: Proceedings of DISCEX. citeseer.ist.psu.edu/portnoy01intrusion.html

Florez-Larrahondo G, Bridges SM, Vaughn R (2005) Efficient modeling of discrete events for anomaly detection using hidden Markov models. Inf Secur 3650:506

Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A sense of self for unix processes. In: Proceedings of the ISRSP'96, pp 120–128. citeseer.ist.psu.edu/forrest96sense.html

Forrest S, Warrender C, Pearlmutter B (1999) Detecting intrusions using system calls: alternate data models. In: Proceedings of the 1999 IEEE ISRSP. IEEE Computer Society, Washington, DC, pp 133–145

Fox AJ (1972) Outliers in time series. J R Stat Soc Ser. B(Methodolog) 34(3):350

Fu AWC, Leung OTW, Keogh EJ, Lin J (2006) Finding time series discords based on haar transform. In: Proceeding of the 2nd International conference on advanced data mining and applications. Springer, Berlin/New York, pp 31–41

Fujimaki R, Yairi T, Machida K (2005) An anomaly detection method for spacecraft using relevance vector learning. In: Proceeding of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining. ACM Press, New York, pp 401–410. doi:http://doi.acm.org/10.1145/1081870.1081917

Fujimaki R, Yairi T, Machida K (2005) An approach to spacecraft anomaly detection problem using kernel feature space. Adv Knowl Discov Data Min 3518:785

Galeano P, Pena D, Tsay RS (2004) Outlier detection in multivariate time series via projection pursuit. Statistics and Econometrics Working Papers ws044211, Universidad Carlos III, Departamento de Estadïstica y Econometrïca

Gao J, Tan PN (2006) Converting output scores from outlier detection algorithms into probability estimates. In: Proceedings of the sixth international conference on data mining (ICDM '06), Hong Kong, pp 212–221

Gao J, Liang F, Fan W, Wang C, Sun Y, Han J (2010) On community outliers and their efficient detection in information networks. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '10), Washington, DC, pp 813–822

Ghosh AK, Schwartzbard A, Schatz M (1999) Learning program behavior profiles for intrusion detection. In: Proceedings of SANS third conference and workshop on intrusion detection and response. citeseer.ist.psu.edu/ghosh99learning.html

Ghosh AK, Schwartzbard A, Schatz M (1999) Using program behavior profiles for intrusion detection. In: Proceedings of 1st USENIX workshop on intrusion detection and network monitoring, Santa Clara, pp 51–62

Hawkins S, He H, Williams GJ, Baxter RA (2002) Outlier detection using replicator neural networks. In: Proceedings of the 4th international conference on data warehousing and knowledge discovery. Springer, Berlin, pp 170–180

He Z, Deng S, Xu X, Huang JZ (2006) A fast greedy algorithm for outlier mining. In: Proceedings of 10th Pacific-Asia conference on knowledge and data discovery, pp 567–576

He Z, Xu X, Deng S (2003) Discovering cluster-based local outliers. Pattern Recognit Lett 24(9–10):1641. doi:http://dx.doi.org/10.1016/S0167-8655(03)00003-5

He Z, Xu X, Deng S (2005) An optimization model for outlier detection in categorical data. In: Proceedings of international conference on intelligent computing, vol 3644. Springer, Berlin/Heidelberg

Hodge V, Austin J (2004) A survey of outlier detection methodologies. Artif Intell Rev 22(2):85. doi:http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

Hofmeyr SA, Forrest S, Somayaji A (1998) Intrusion detection using sequences of system calls. J Comput Secur 6(3):151. citeseer.ist.psu.edu/hofmeyr98intrusion.html

Keogh E, Lin J, Fu A (2005) Hot sax: Efficiently finding the most unusual time series subsequence. In: Proceedings of the fifth IEEE international conference on data mining, IEEE Computer Society, Washington, DC, pp 226–233. doi:http://dx.doi.org/10.1109/ICDM.2005.79

Keogh E, Lin J, Lee SH, Herle HV (2006) Finding the most unusual time series subsequence: algorithms and applications. Knowl Inf Syst 11(1):1. doi:http://dx.doi.org/10.1007/s10115-006-0034-6

Keogh E, Lonardi S, Ratanamahatana CA (2004) Towards parameter-free data mining. In: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, New York, pp 206–215. doi:http://doi.acm.org/10.1145/1014052.1014077

Khoa NLD, Chawla S (2010) Robust outlier detection using commute time and eigenspace embedding. In: Advances in knowledge discovery and data mining, 14th Pacific-Asia conference, PAKDD 2010. Proceedings, Part II. Hyderabad, 21–24 June 2010, pp 422–434

Knorn F, Leith D (2008) Adaptive Kalman filtering for anomaly detection in software appliances. In: IEEE INFOCOM workshops 2008, Phoenix, AZ, pp 1–6

Knorr EM, Ng RT (1997) A unified approach for mining outliers. In: Proceedings of the 1997 conference of the centre for advanced studies on collaborative research. IBM Press, Toronto, p 11

Knorr EM, Ng RT (1999) Finding intensional knowledge of distance-based outliers. In: The VLDB journal, pp 211–222. citeseer.ist.psu.edu/knorr99finding.html

Knorr EM, Ng RT, Tucakov V (2000) Distance-based outliers: algorithms and applications. VLDB J 8(3–4):237. doi:http://dx.doi.org/10.1007/s007780050006

Kou Y, Lu CT, Chen D (2006) Spatial weighted outlier detection. In: Proceedings of SIAM conference on data mining, Bethesda

Kriegel HP, Hubert MS, Zimek A (2008) Angle-based outlier detection in highdimensional data. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '08), Las Legas, pp 444–452

Kriegel HP, Krger P, Schubert E, Zimek A (2011) Interpreting and unifying outlier scores. In: SDM. SIAM/Omnipress, Mesa, AZ, USA, pp 13–24

Kumar V (2005) Parallel and distributed computing for cybersecurity. Distributed systems online. IEEE 6(10). doi:10.1109/MDSO.2005.53

Lane T, Brodley CE (1997) Sequence matching and learning in anomaly detection for computer security. In: Fawcett T, Haimowitz I, Provost F, Stolfo S (eds) Proceedings of AI approaches to fraud detection and risk management. AAAI Press, Menlo Park, pp 43–49

Lane T, Brodley CE (1999) Temporal sequence learning and data reduction for anomaly detection. ACM Trans Inf Syst Secur 2(3):295. doi:http://doi.acm.org/10.1145/322510.322526

Laurikkala J, Juhola1 M, Kentala E (2000) Informal identification of outliers in medical data. In: Fifth international workshop on intelligent data analysis in medicine and pharmacology, Berlin, pp 20–24

Lee W, Xiang D (2001) Information-theoretic measures for anomaly detection. In: Proceedings of the IEEE symposium on security and privacy. IEEE Computer Society, Washington, DC, p 130

Lee W, Stolfo S, Chan P (1997) Learning patterns from unix process execution traces for intrusion detection. In: Proceedings of the AAAI 97 workshop on AI methods in fraud and risk management

Li N, Sun H, Chipman KC, George J, Yan X (2014) A probabilistic approach to uncovering attributed graph anomalies. In: Proceedings of the 2014 SIAM international conference on data mining, Philadelphia, pp 82–90, 24–26 Apr 2014. doi:10.1137/1.9781611973440.10, http://dx.doi.org/10.1137/1.9781611973440.10

Lin J, Keogh E, Fu A, Herle HV (2005) Approximations to magic: finding unusual medical time series. In: Proceedings of the 18th IEEE symposium on computer-based medical systems. IEEE Computer

Society, Washington, DC, pp 329–334. doi:http://dx.doi.org/10.1109/CBMS.2005.34

Ma J, Perkins S (2003) Online novelty detection on temporal sequences. In: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, New York, pp 613–618. doi:http://doi.acm.org/10.1145/956750.956828

Mahoney MV, Chan PK (2002) Learning nonstationary models of normal network tra c for detecting novel attacks. In: Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, pp 376–385. doi:http://doi.acm.org/10.1145/775047.775102

Mahoney MV, Chan PK (2003) Learning rules for anomaly detection of hostile network traffic. In: Proceedings of the 3rd IEEE international conference on data mining. IEEE Computer Society, Los Alamitos, p 601

Mahoney MV, Chan PK (2005) Trajectory boundary modeling of time series for anomaly detection. In: Proceedings of the KDD workshop on data mining methods for anomaly detection, Las Vegas, NV, USA

Mahoney MV, Chan PK, Arshad MH (2003) A machine learning approach to anomaly detection. Technical report CS–2003–06, Department of Computer Science, Florida Institute of Technology Melbourne, FL, 32901

Marchette D (1999) A statistical method for profiling network traffic. In: Proceedings of 1st USENIX workshop on intrusion detection and network monitoring, Santa Clara, pp 119–128

Michael CC, Ghosh A (2000) Two state-based approaches to program-based anomaly detection. In: Proceedings of the 16th annual computer security applications conference, IEEE Computer Society, Los Alamitos, p 21

Müller E, Sanchez PI, Mülle Y, Böhm K (2013) Ranking outlier nodes in subspaces of attributed graphs. In: Workshops proceedings of the 29th IEEE international conference on data engineering. ICDE, pp 216–222

Noble CC, Cook DJ (2003) Graph-based anomaly detection. In: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, pp 631–636. doi:http://doi.acm.org/10.1145/956750.956831

Otey ME, Ghoting A, Parthasarathy S (2006) Fast distributed outlier detection in mixed-attribute data sets. Data Min Knowl Discov 12(2–3):203. doi:http://dx.doi.org/10.1007/s10618-005-0014-6

Ott L, Pang LX, Ramos FT, Chawla S (2014) On integrated clustering and outlier detection. In: Advances in neural information processing systems, pp 1359–1367

Parra L, Deco G, Miesbach S (1996) Statistical independence and novelty detection with information preserving nonlinear maps. Neural Comput 8 (2):260

Pincombe B (2005) Anomaly detection in time series of graphs using ARMA processes. ASOR Bull 24(4):2

Pokrajac D, Lazarevic A, Latecki LJ (2007) Incremental local outlier detection for data streams. In: Proceedings of IEEE symposium on computational intelligence and data mining

Portnoy L, Eskin E, Stolfo S (2001) Intrusion detection with unlabeled data using clustering. In: Proceedings of ACM workshop on data mining applied to security. citeseer.ist.psu.edu/portnoy01intrusion.html

Protopapas P, Giammarco JM, Faccioli L, Struble MF, Dave R, Alcock C (2006) Finding outlier light curves in catalogues of periodic variable stars. Mon Notices R Astron Soc 369(2):677

Qiao Y, Xin XW, Bin Y, Ge S (2002) Anomaly intrusion detection method based on HMM. Electron Lett 38(13):663

Ramaswamy S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large data sets. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. ACM Press, New York, pp 427–438. doi:http://doi.acm.org/10.1145/342009.335437

Roth V (2004) In: NIPS

Rousseeuw PJ, Leroy AM (1987) Robust regression and outlier detection. Wiley, New York

Salvador S, Chan P (2003) Learning states and rules for time-series anomaly detection. Technical report CS–2003–05, Department of Computer Science, Florida Institute of Technology Melbourne FL 32901

Salvador S, Chan P (2005) Learning states and rules for detecting anomalies in time series. Appl Intell 23(3):241. doi:http://dx.doi.org/10.1007/s10489-005-4610-3

Sánchez PI, Müller E, Irmler O, Böhm K (2014) Local context selection for outlier ranking in graphs with multiple numeric node attributes. In: Proceedings of the 26th International conference on scientific and statistical database management (SSDBM '14). ACM, New York, pp 16:1–16:12. doi:10.1145/2618243.2618266. http://doi.acm.org/10.1145/2618243.2618266

Schölkopf B, Platt JC, Shawe-Taylor JC, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443

Schubert E, Wojdanowski R, Zimek A, Kriegel HP (2012) In: SDM. SIAM/Omnipress, Anaheim, CA, USA, pp 1047–1058

Shekhar S, Lu CT, Zhang P (2001) A novel anomaly detection scheme based on principal component classifier. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, New York, pp 371–376. doi:http://doi.acm.org/10.1145/502512.502567

Shyu ML, Chen SC, Sarinnapakorn K, Chang L (2003) A novel anomaly detection scheme based on principal component classifier. In: Proceedings of 3rd

IEEE international conference on data mining, Melbourne, pp 353–365
Song X, Wu M, Jermaine C, Ranka S (2007) Conditional anomaly detection. IEEE Trans Knowl Data Eng 19(5):631 doi:http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.1009
Spence C, Parra L, Sajda P (2001) Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. In: Proceedings of the IEEE workshop on mathematical methods in biomedical image analysis. IEEE Computer Society, Washington, DC, p 3
Sun J, Qu H, Chakrabarti D, Faloutsos C (2005) Relevance search and anomaly detection in bipartite graphs. SIGKDD Explor Newslett 7(2):48
Sun P, Chawla S, Arunasalam B (2006) Mining for outliers in sequential databases. In: SIAM international conference on data mining, Philadelphia
Tan PN, Steinbach M, Kumar V (2005) Introduction to data mining. Addison-Wesley, Boston
Tang J, Chen Z, chee Fu AW, Cheung DW (2002) Enhancing effectiveness of outlier detections for low density patterns. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, Taipei, pp 535–548
Tax DMJ (2001) One-class classification; concept-learning in the absence of counter-examples. PhD thesis, Delft University of Technology
Tax D, Duin R (1999) Data domain description using support vectors. In: Verleysen M (ed) Proceedings of the European symposium on artificial neural networks, Brussels, pp 251–256
Tax D, Duin R (1999) Support vector data description. Pattern Recognit Lett 20(11–13):1191
Tong H, Lin C-Y (2011) Non-negative residual matrix factorization with application to graph anomaly detection. In: Proceedings of the 2011 SIAM international conference on data mining, Philadelphia, pp 143–153
Tsay RS, Peja D, Pankratz AE (2000) Outliers in multivariate time series. Biometrika 87(4):789
Vaidya J, Clifton C (2004) Privacy-preserving outlier detection. In: Proceedings of the 4th IEEE international conference on data mining, Brighton, pp 233–240
Wei L, Keogh E, Xi X (2006) Saxually explicit images: Finding unusual shapes. In: Proceedings of the sixth international conference on data mining, IEEE Computer Society, Washington, DC, pp 711–720. doi:http://dx.doi.org/10.1109/ICDM.2006.138
Wei L, Kumar N, Lolla V, Keogh EJ, Lonardi S, Ratanamahatana C (2005) Assumption-free anomaly detection in time series. In: Proceedings of the 17th international conference on Scientific and statistical database management, Lawrence Berkeley Laboratory, Berkeley, pp 237–240
Weigend AS, Mangeas M, Srivastava AN (1995) Nonlinear gated experts for timeseries – discovering regimes and avoiding overfitting. Int J Neural Syst 6(4):373
Wu Q, Shao Z (2005) Network anomaly detection using time series analysis. In: Proceedings of the joint international conference on autonomic and autonomous systems and international conference on networking and services. IEEE Computer Society, Washington, DC, p 42
Yang J, Wang W (2003) CLUSEQ: Efficient and effective sequence clustering. In: Proceedings of international conference on data engineering, Bangalore, pp 101–112
Yankov D, Keogh EJ, Rebbapragada U (2007) Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In: Proceedings of international conference on data mining, pp 381–390
Ye N (2004) A Markov Chain model of temporal behavior for anomaly detection. In: Proceedings of the 5th annual IEEE information assurance workshop. IEEE, Piscataway
Zare Moayedi H, Masnadi-Shirazi M (2008) ARIMA model for network traffic prediction and anomaly detection. Int Symp Inf Technol 4:1. doi:10.1109/ITSIM.2008.4631947
Zhang J, Wang H (2006) Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. Knowl Inf Syst 10(3):333. doi:http://dx.doi.org/10.1007/s10115-006-0020-z
Zhang X, Fan P, Zhu Z (2003) A new anomaly detection method based on hierarchical HMM. In: Proceedings of the 4th international conference on parallel and distributed computing, applications and technologies, Chengdu, pp 249–252
Zimmermann J, Mohay G (2006) Distributed intrusion detection in clusters based on non-interference. In: ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on grid computing and e-research. Australian Computer Society, Darlinghurst, pp 89–95

Ant Colony Optimization

Marco Dorigo and 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 on-line 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 et al. (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 $x_k = \langle c_1, c_2, \ldots, c_k \rangle \in \mathcal{X}, x_k \notin \mathcal{S}$, 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 α, $\beta > 0$, 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 and 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, $\rho, 0 \leq \rho < 1$, 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 et al. 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 and 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 and 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 and 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 and 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 JL, Goss S (1992) Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. J Theor Biol 159:397–415

Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66

Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge

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

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern – Part B 26(1): 29–41

Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369

Stützle T, Hoos HH (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. IEEE Press, Piscataway, pp 309–314

Stützle T, Hoos HH (2000) $\mathcal{MAX}-\mathcal{MIN}$ ant system. Future Gener Comput Syst 16(8):889–914

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 et al. 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 : = ∅;
2: candidate_sets : = $\{\{i\} | i \in \mathcal{I}\}$;
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 $| \text{frequency}(C) \geq min\_fr\}$;
    3.3: level := level + 1;
    3.4: candidate_sets := $\{A \subset \mathcal{I} \,||\, A| =$ 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 AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328

AQ

► Rule Learning

Architecture

► Topology of a Neural Network

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.

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 and 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 et al. 1986; Varela et al. 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 (Cooke and 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 (Cooke and 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 (2005) (later expanded in Watkins 2005) developed an immune-based classifier known as AIRS, and in the dynamic supervised domain, with the work in Secker et al. (2003) being one of a number of successes.

Structure of the Learning System

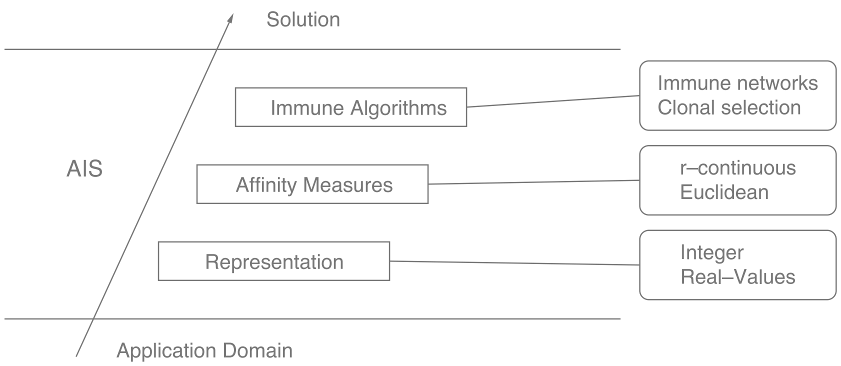
In an attempt to create a common basis for AIS, the work in de Castro and Timmis (2002) proposed the idea of a framework for engineering AIS. They argued that the case for such a framework as the existence of similar frameworks in other biologically inspired approaches, such as ▸ artificial neural networks (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 and 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 and 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



Artificial Immune Systems, Fig. 1 AIS layered framework (Adapted from de Castro and Timmis 2002) of the system are standard in 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 and 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)

   (d) Combine the elements of C with the set P
   (e) 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). MIT Press, Cambridge, pp 217–226

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

de Castro LN, Timmis J (2002) Artificial immune systems: a new computational intelligence approach. Springer, New York

de Castro LN, Von Zuben FJ (2001) aiNet: an artificial immune network for data analysis. Idea Group Publishing, Hershey, pp 231–259

Farmer JD, Packard NH, Perelson AS (1986) The immune system, adaptation, and machine learning. Physica D 22:187–204

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

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

Hart E, Timmis J (2008) Application areas of AIS: the past, present and the future. J Appl Soft Comput 8(1):191–201

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

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

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

Timmis J, Neal M (2001) A resource limited artificial immune system for data analysis. Knowl Based Syst 14(3–4):121–130

Varela F, Coutinho A, Dupire B, Vaz N (1988) Cognitive networks: immune, neural and otherwise. J Theor Immunol 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. Ph.D. 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
▶ Topology of a Neural Network
▶ Neuroevolution
▶ Radial Basis Function Networks
▶ Reservoir Computing
▶ Self-Organizing Maps
▶ Simple Recurrent Network
▶ Weight

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 and Axtell 1996), and the Santa Fe artificial stock market (Arthur et al. 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 (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
▸ Coevolutionary Learning
▸ Multi-agent Learning

Recommended Reading

Agent-based computational economics, website maintained by Tesfatsion (2009)
Arifovic J (1994) Genetic algorithm learning and the cobweb-model. J Econ Dyn Control 18:3–28
Arthur B, Holland J, LeBaron B, Palmer R, Taylor P (1997) Asset pricing under endogenous expecta-

tions in an artificial stock market. In: Arthur B et al. (eds) The economy as an evolving complex system II. Addison-Wesley, Boston, pp 5–44
Axelrod: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration (Axelrod 1997)
Axelrod R (1997) The complexity of cooperation: agent-based models of competition and collaboration. Princeton University Press, Princeton
Axtell RL, Epstein JM, Dean JS, Gumerman GJ, Swedlund AC, Harburger J et al (2002) Population growth and collapse in a multiagent model of the kayenta anasazi in long house valley. Proc Natl Acad Sci 99:7275–7279
Bonabeau: Agent-based modeling (Bonabeau 2002)
Brenner: Agent learning representation: Advice on modeling economic learning (Brenner 2006)
Bonabeau E (2002) Agent-based modeling: methods and techniques for simulating human systems. Proc Natl Acad Sci 99:7280–7287
Brenner T (2006) Agent learning representation: advice on modelling economic learning. In: Tesfatsion L, Judd KL (eds) Handbook of computational economics, vol 2. North-Holland, Amsterdam, pp 895–947
Duffy J (2006) Agent-based models and human subject experiments. In: Tesfatsion L, Judd KL (eds) Handbook of computational economics, vol 2. North-Holland, Amsterdam, pp 949–1011
Epstein: Generative social science (Epstein 2006)
Epstein JM (2006) Generative social science: studies in agent-based computational modeling. Princeton University Press, Princeton
Epstein JM, Axtell R (1996) Growing artificial societies. Brookings Institution Press, Washington, DC
Gilbert N, den Besten M, Bontovics A, Craenen BGW, Divina F, Eiben AE et al (2006) Emerging artificial societies through learning. J Artif Soc Soc Simul 9(2). http://jasss.soc.surrey.ac.uk/9/2/9.html
Journal of Artificial Societies and Social Simulation (2009)
Railsback SF, Lytinen SL, Jackson SK (2006) Agent-based simulation platforms: review and development recommendations. Simulation, 82(9):609–623
Schelling TC (1969) Dynamic models of segregation. J Math Soc 2:143–186
Tesfatsion and Judd (eds.): Handbook of computational economics (Tesfatsion and Judd 2006)
Tesfatsion L (2009) Website on agent-based computational economics. http://www.econ.iastate.edu/tesfatsi/ace.htm
Tesfatsion L, Judd KL (eds) (2006a) Handbook of computational economics. Elsevier, Amsterdam/Oxford
Tesfatsion L, Judd KL (eds) (2006b) Handbook of computational economics – vol 2: agent-based computational economics. Elsevier, Amsterdam
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.

Assessment of Model Performance

▸ Model Evaluation

Association Rule

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Association rules (Agrawal et al. 1993) can be extracted from data sets where each example consists of a set of items. An association rule has the form $X \to Y$, 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 $X \to Y$, denoted $\mathrm{fr}(X \to Y)$, 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 $P(Y|X) = \mathrm{fr}(X \cup Y)/\mathrm{fr}(X)$.

The ▸ Apriori algorithm (Agrawal et al. 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, Imieliñski 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. ACM, New York, pp 207–216

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

Associative Bandit Problem

▶ Associative Reinforcement Learning

Associative Reinforcement Learning

Alexander L. Strehl
Rütgers University, New Brunswick, NJ, 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 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

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

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 et al. (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 (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) and Fiechter (PAC associative reinforcement learning, unpublished manuscript, 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 et al. (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, 1990), ARC (Sutton, 1984), and REINFORCE (Williams, 1992)

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

Ackley DH, Littman ML (1990) Generalization and scaling in reinforcement learning. In: Advances in neural information processing systems 2. Morgan Kaufmann, San Mateo, pp 550–557

Auer P (2002) Using confidence bounds for exploitation–exploration trade-offs. J Mach Learn Res 3:397–422

Kaelbling LP (1994) Associative reinforcement learning: functions in k-DNF. Mach Learn 15:279–298

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285

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

Sutton RS (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, University of Massachusetts, Amherst

Valiant LG (1984) A theory of the learnable. Commun ACM 27:1134–1142

Wang C-C, Kulkarni SR, Poor HV (2005) Bandit problems with side observations. IEEE Trans Autom Control 50:3988–3993

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 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 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 and 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 and Kramer 2000; Dietterich et al. 2004; Fern et al. 2006).

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.

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, Bonn

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

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

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

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

Quinlan JR (1996) Learning first-order definitions of functions. J Artif Intell Res 5:139–161

Simoff SJ, Djeraba C (eds) (2000) Workshop on multimedia data mining. In: Proceedings of the sixth international conference on knowledge discovery and data mining, Boston

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

Authority Control

▶ Record Linkage

Autonomous Helicopter Flight Using Reinforcement Learning

Adam Coates[1], Pieter Abbeel[2], and Andrew Y. Ng[1,3]
[1]Stanford University, Stanford, CA, USA
[2]EECS Department, UC Berkeley, Stanford, CA, USA
[3]Computer Science Department, 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, the 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 and Schneider 2001; Ng et al. 2004b), 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 and Ng 2004). These methods have enabled autonomous helicopters to fly aerobatics as well as an expert human pilot and often even better (Coates et al. 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 are 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 12-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 onboard 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 maintains 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 et al. 2007; Coates et al. 2008; Saripalliz et al. 2003).

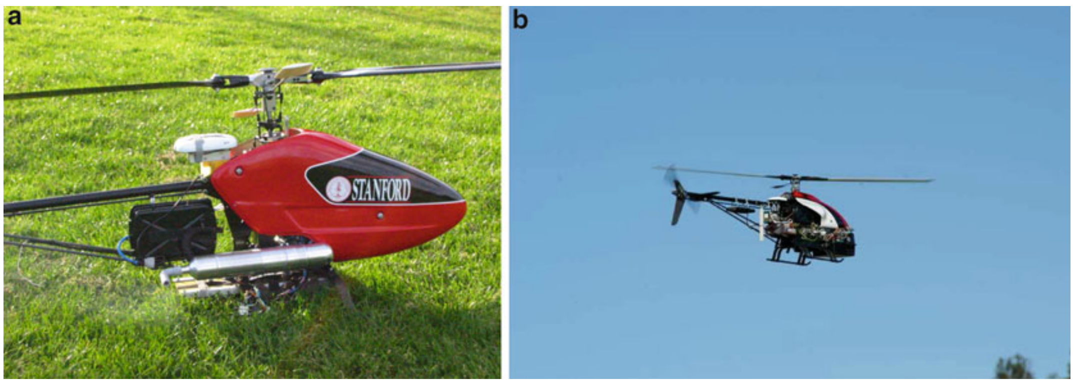
Vibration adds errors to the sensor measurements and may damage the sensors themselves; hence, significant effort may be required to mount the sensors on the airframe (Dunbabin et al. 2004). Provided there is no aliasing, sensor errors added by vibration can be removed by using a digital filter on the 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 onboard computer that runs the EKF and the control algorithm (Gavrilets et al. 2002a; La Civita et al. 2006; Ng et al. 2004a). 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



Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 1 (**a**) Stanford University's instrumented XCell Tempest autonomous helicopter. (**b**) A Bergen Industrial Twin autonomous helicopter with sensors and onboard computer 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

An 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 and 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 et al. 2001, 2002b; 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 and 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, nonparametric algorithms such as locally weighted linear regression have been used (Bagnell and Schneider 2001; Ng et al. 2004b). Nonparametric 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 nonparametric 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 et al. 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. Abbeel et al. (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) and Ng et al. (2004b)). 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 and Ng 2005b).

The models considered above are deterministic. This normally would allow us to drop the expectation when evaluating a policy according to $\mathrm{E}\left[\sum_{t=0}^{H} R(s(t), u(t))|\pi\right]$. However, it is common to add stochasticity to account for unmodeled effects. Abbeel et al. (2007) and Ng et al. (2004a) 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 $U(\pi) = \mathrm{E}\left[\sum_{t=0}^{H} R(s(t), u(t))|\pi\right]$ 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 and 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. (2004a) 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 and 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. (2004a), 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 and 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 and 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 et al. (2008), Fig. 2.

While DDP computes a solution to the nonlinear optimization problem, it relies on the accuracy of the nonlinear 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 nonlinear 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 and the trade-offs between different types of errors. If the desired trajectory is infeasible (either in the nonlinear 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

Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 2 Snapshots of an autonomous helicopter performing in-place flips and rolls 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 and 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, $\{s_k(t), u_k(t)\}_{t=0}^{H}$ for $k = 1, \ldots, N$, from demonstration flights by an expert pilot. Coates et al. (2008) note that these demonstrations may be suboptimal but are often suboptimal 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 et al. 1977) algorithm, they infer the desired trajectory and use this as the target trajectory in their reward function.

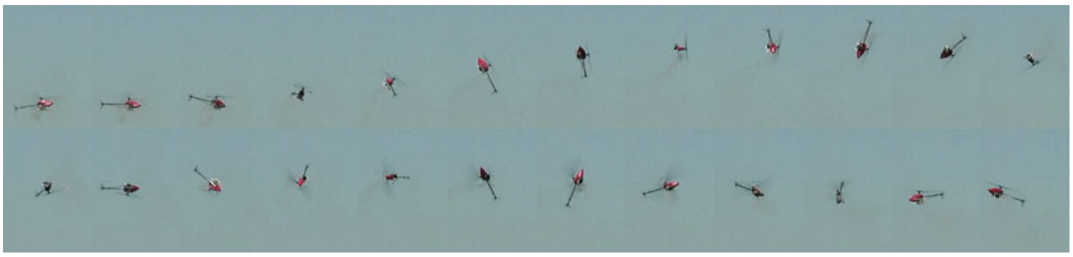
A good choice of reward weights (for errors relative to the desired trajectory) can be recovered using inverse reinforcement learning (Ng and Russell 2000; Abbeel and Ng 2004). Suppose the reward function is written as a linear combination of features as follows: $R(s,u) = c_0\phi_0(s,u)+c_1\phi_1(s,u)+\cdots$. 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 outperforms 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 and Singh 2002), this procedure does not require explicit exploration policies. One only needs to test controllers that try to fly as much as possible (according to the current choice of dynamics model). (Indeed, the E^3-family of algorithms (Kearns and Singh 2002) and its extensions (Kearns and Koller 1999; Brafman and Tennenholtz 2002; Kakade et al. 2003) 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 ex-



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

pert level performance on challenging aerobatic maneuvers as well as entire air shows 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 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 air shows 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

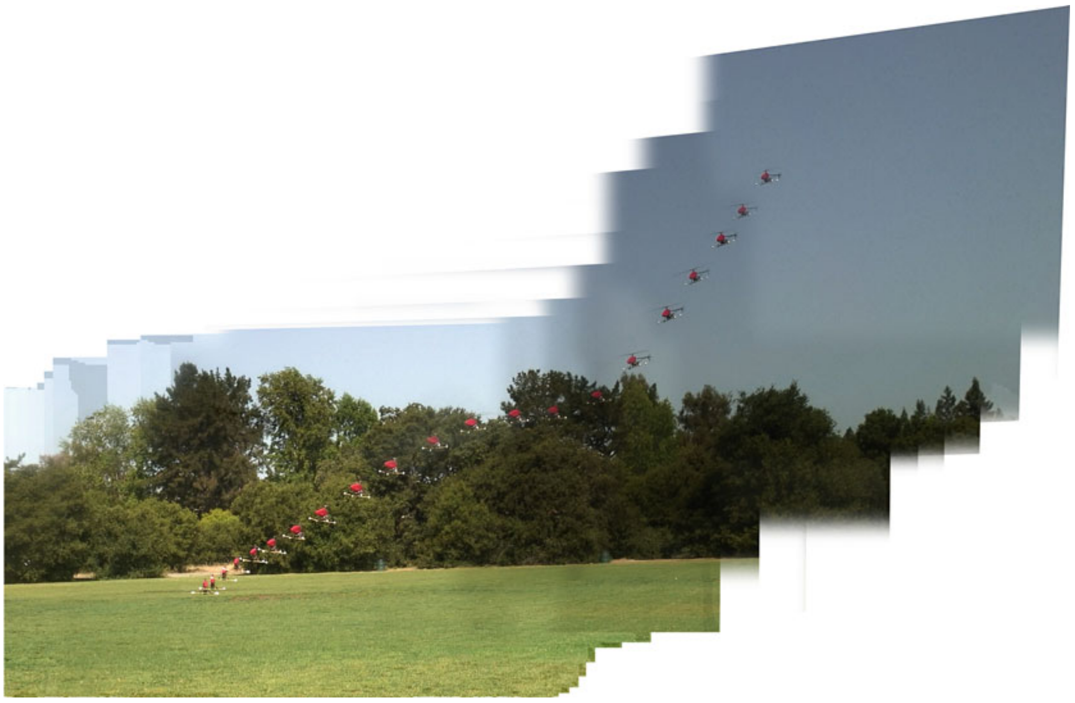
Helicopter control is a challenging control problem and has recently seen major successes with the application of learning algorithms. This entry has shown how each step of the control design process can be automated using machine learning algorithms for system identification and reinforcement learning 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 autorotation) at the level of expert human pilots.

Cross-References

▶ Apprenticeship Learning
▶ Reinforcement Learning
▶ Reward Shaping

Recommended Reading

Abbeel P, Coates A, Hunter T, Ng AY (2008) Autonomous autorotation of an rc helicopter. In: ISER 11, Athens



Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 4 Superimposed sequence of images of autonomous autorotation landings (from Abbeel et al. 2008)

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

Abbeel P, Ganapathi V, Ng AY (2006) Learning vehicular dynamics with application to modeling helicopters. In: NIPS 18, Vancouver

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

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

Abbeel P, Ng AY (2005b) Learning first order Markov models for control. In: NIPS 18, Vancouver

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

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

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

Brafman RI, Tennenholtz M (2002) R-max, a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3: 213–231

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

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 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, New Orleans, 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, Monterey. Massachusetts Institute of Technology, Cambridge

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, Irvine

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

Jacobson DH, Mayne DQ (1970) Differential dynamic programming. Elsevier, New York

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

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

Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn J 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

La Civita M, Papageorgiou G, Messner WC, Kanade T (2006) Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. J Guid Control Dyn 29(2):485–494

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

Nelder JA, Mead R (1964) A simplex method for function minimization. Comput J 7:308–313

Ng AY, 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, Singapore. Springer, Berlin

Ng AY, Jordan M (2000) PEGASUS: a policy search method for large MDPs and POMDPs. In: Proceedings of the uncertainty in artificial intelligence 16th conference, Stanford. Morgan Kaufmann, San Francisco

Ng AY, Kim HJ, Jordan M, Sastry S (2004) Autonomous helicopter flight via reinforcement learning. In: NIPS 16, Vancouver

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

Saripalli S, Montgomery JF, Sukhatme GS (2003) Visually-guided landing of an unmanned aerial vehicle. IEEE Trans Robot Auton Syst 19(3):371–380

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

Tischler MB, Cauffman MG (1992) Frequency response method for rotorcraft system identification: flight application to BO-105 couple rotor/fuselage dynamics. J Am Helicopter Soc 37:3–17

Average-Cost Neuro-Dynamic Programming

▶ Average-Reward Reinforcement Learning

Average-Cost Optimization

▶ Average-Reward Reinforcement Learning

Averaged One-Dependence Estimators

Fei Zheng[1,2] and Geoffrey I. Webb[3]
[1]Monash University, Syndey, NSW, Australia
[2]Monash University, Victoria, Australia
[3]Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

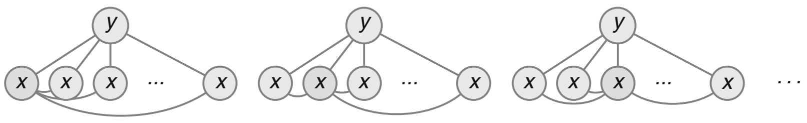
AODE

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.



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

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,

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

This equality holds for every x_i. Therefore,

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

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

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

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 et al. 2005; Zheng and 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 et al. 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.

In more recent work (Webb et al. 2012), AODE has been generalized to Averaged N-Dependence Estimators (ANDE) and it has been demonstrated that bias can be further decreased by introducing multiple SuperParents to each submodel.

Cross-References

▸ Bayesian Network
▸ Naïve Bayes
▸ Semi-Naive Bayesian Learning
▸ Tree Augmented Naive Bayes

Recommended Reading

Webb GI, Boughton J, Wang Z (2005) Not so naive Bayes: aggregating one-dependence estimators. Mach Learn 58(1):5–24

Webb GI, Boughton J, Zheng F, Ting KM, & Salem H (2012) Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. Mach Learn 86(2): 233–272.

Zheng F, Webb GI (2005) A comparative study of semi-naive Bayes methods in classification learning. In: Proceedings of the fourth Australasian data mining conference, Sydney, pp 141–156

Average-Payoff Reinforcement Learning

▶ Average-Reward Reinforcement Learning

Average-Reward Reinforcement Learning

Prasad Tadepalli
School of Electrical Engineering and Computer Science, 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.

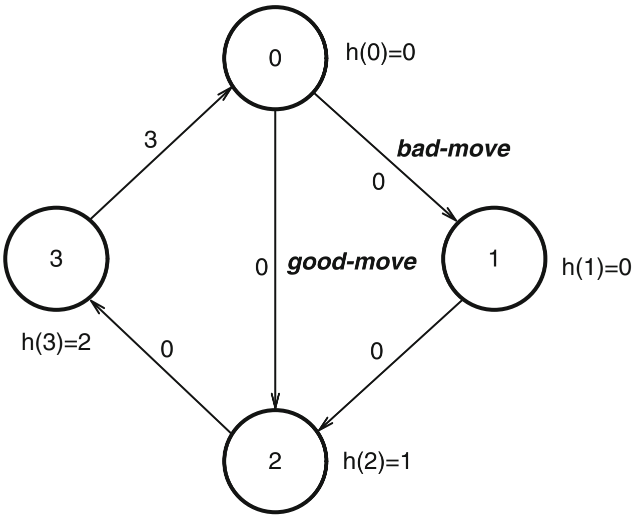
Motivation and Background

▶ Reinforcement learning (RL) is the study of programs that improve their performance at some task by receiving rewards and punishments from the environment (Sutton and Barto 1998). RL has been quite successful in the automatic learning of good procedures for complex tasks such as playing Backgammon and scheduling elevators (Tesauro 1992; Crites and Barto 1998). 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 a 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



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

need to be learned for optimal performance. The following theorem gives the Bellman equation for the bias values of states.

Theorem 1 *For ergodic MDPs, there exist a scalar ρ and a real-valued bias function h over S that satisfy the recurrence relation*

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

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.

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 and Ok 1998). This is an average-reward version of ▸ Adaptive real-time dynamic programming (Barto et al. 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.

$$\rho \leftarrow \rho + \alpha(r - h(s) + h(s')). \tag{3}$$

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.

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

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:

$$h(s') = \max_u R(s',u). \tag{5}$$

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

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

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 and 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 et al. 2000), local ▶ linear regression (Tadepalli and Ok 1998), and tile coding (Proper and 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 the 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 and Tadepalli 2002) and model-free approaches (Ghavamzadeh and 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 and Mahadevan 2006) and transfer line optimization (Wang and 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 et al. 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 and 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 and 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 and 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 and 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 DP, Borkar V (2002) Stochastic approximation for non-expansive maps: application to Q-learning algorithms. SIAM J Control Optim 41(1):1–22

Barto AG, Bradtke SJ, Singh SP (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1):81–138

Bertsekas DP (1995) Dynamic programming and optimal control. Athena Scientific, Belmont

Brafman RI, Tennenholtz M (2002) R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 2:213–231

Crites RH, Barto AG (1998) Elevator group control using multiple reinforcement agents. Mach Learn 33(2/3):235–262

Ghavamzadeh M, Mahadevan S (2006) Hierarchical average reward reinforcement learning. J Mach Learn Res 13(2):197–229

Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2/3):209–232

Mahadevan S (1996) Average reward reinforcement learning: foundations, algorithms, and empirical results. Mach Learn 22(1/2/3):159–195

Marbach P, Mihatsch O, Tsitsiklis JN (2000) Call admission control and routing in integrated service networks using neuro-dynamic programming. IEEE J Sel Areas Commun 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, Berlin. Springer, pp 725–742

Puterman ML (1994) Markov decision processes: discrete dynamic stochastic programming. Wiley, New York

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

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

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT, Cambridge

Tadepalli P, Ok D (1998) Model-based average-reward reinforcement learning. Artif Intell 100:177–224

Tesauro G (1992) Practical issues in temporal difference learning. Mach Learn 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. Mach Learn 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, Bled, pp 464–473