doi:10.1038/nature14540

# Reinforcement learning improves behaviour from evaluative feedback

Michael L. Littman[1]

**Reinforcement learning is a branch of machine learning concerned with using experience gained through interacting with the world and evaluative feedback to improve a system's ability to make behavioural decisions. It has been called the artificial intelligence problem in a microcosm because learning algorithms must act autonomously to perform well and achieve their goals. Partly driven by the increasing availability of rich data, recent years have seen exciting advances in the theory and practice of reinforcement learning, including developments in fundamental technical areas such as generalization, planning, exploration and empirical methodology, leading to increasing applicability to real-life problems.**

Reinforcement-learning algorithms[1,2] are inspired by our understanding of decision making in humans and other animals in which learning is supervised through the use of reward signals in response to the observed outcomes of actions. As our understanding of this class of problems improves, so does our ability to bring it to bear in practical settings. Reinforcement learning is having a considerable impact on nuts-and-bolts questions such as how to create more effective personalized Web experiences, as well as esoteric questions such as how to design better computerized players for the traditional board game Go or 1980s video games. Reinforcement learning is also providing a valuable conceptual framework for work in psychology, cognitive science, behavioural economics and neuroscience that seeks to explain the process of decision making in the natural world.

One way to think about machine learning is as a set of techniques to try to answer the question: when I am in situation $x$, what response should I choose? As a concrete example, consider the problem of assigning patrons to tables in a restaurant. Parties of varying sizes arrive in an unknown order and the host allocates each party to a table. The host maps the current situation $x$ (size of the latest party and information about which tables are occupied and for how long) to a decision (which table to assign to the party), trying to satisfy a set of competing goals such as minimizing the waiting time for a table, balancing the load among the various servers, and ensuring that the members of a party can sit together. Similar allocation challenges come up in games such as Tetris and computational problems such as data-centre task allocation. Viewed more broadly, this framework fits any problem in which a sequence of decisions needs to be made to maximize a scoring function over uncertain outcomes.

The kinds of approaches needed to learn good behaviour for the patron-assignment problem depend on what kinds of feedback information are available to the decision maker while learning (Fig. 1).

Exhaustive versus sampled feedback is concerned with the coverage of the training examples. A learner given exhaustive feedback is exposed to all possible situations. Sampled feedback is weaker, in that the learner is only provided with experience of a subset of situations. The central problem in classic supervised learning is generalizing from sampled examples.

Supervised versus evaluative feedback is concerned with how the learner is informed of right and wrong answers. A requirement for applying supervised learning methods is the availability of examples with known optimal decisions. In the patron-assignment problem, a host-in-training could work as an apprentice to a much more experienced supervisor to learn how to handle a range of situations. If the apprentice can only learn from supervised feedback, however, she would have no opportunities to improve after the apprenticeship ends.

By contrast, evaluative feedback provides the learner with an assessment of the effectiveness of the decisions that she made; no information is available on the appropriateness of alternatives. For example, a host might learn about the ability of a server to handle unruly patrons by trial and error: when the host makes an assignment of a difficult customer, it is possible to tell whether things went smoothly with the selected server, but no direct information is available as to whether one of the other servers might have been a better choice. The central problem in the field of reinforcement learning is addressing the challenge of evaluative feedback.

One-shot versus sequential feedback is concerned with the relative timing of learning signals. Evaluative feedback can be subdivided into whether it is provided directly for each decision or whether it has longer-term impacts that are evaluated over a sequence of decisions. For example, if a host needs to seat a party of 12 and there are no large tables available, some past decision to seat a small party at a big table might be to blame. Reinforcement learners must solve this temporal credit assignment problem to be able to derive good behaviour in the face of this weak sequential feedback.

From the beginning, reinforcement-learning methods have contended with all three forms of weak feedback simultaneously — sampled, evaluative and sequential feedback. As such, the problem is considerably harder than that of supervised learning. However, methods that can learn from weak sources of feedback are more generally applicable and can be mapped to a variety of naturally occurring problems, as suggested by the patron-assignment problem.

The following sections describe recent advances in several sub-areas of reinforcement learning that are expanding its power and applicability.

## Bandit problems

The $k$-armed bandit problem concerns learning to make decisions from one-shot, exhaustive evaluative feedback[3]. It is both harder (evaluative versus supervised feedback) and easier (exhaustive versus sampled feedback) than supervised learning. Initially, the problem was motivated by decision making in medical trials — prescribing a treatment to a test subject only reveals the outcome of that experiment and not the outcome of competing treatments. Decision making in this setting involves

[1]Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

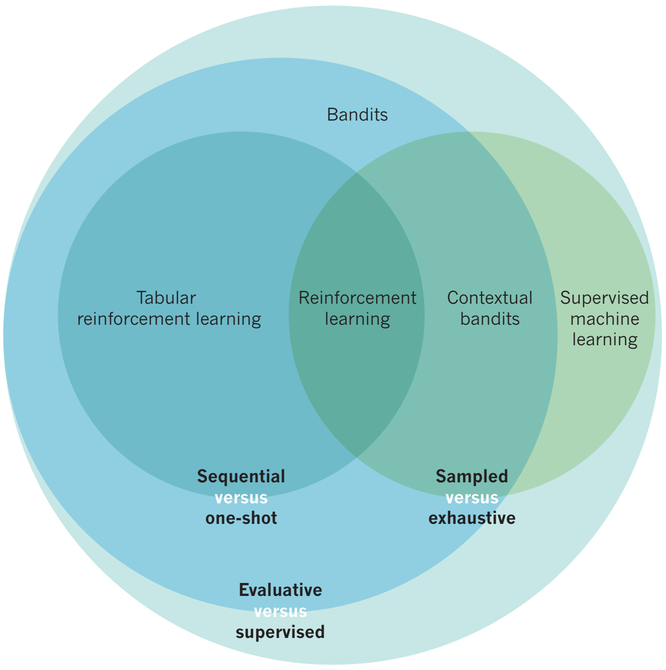

Bandits
Tabular reinforcement learning
Reinforcement learning
Contextual bandits
Supervised machine learning
Sequential versus one-shot
Sampled versus exhaustive
Evaluative versus supervised

**Figure 1 | Decisions of machine-learning feedback.** Three kinds of feedback (sequential versus one-shot, sampled versus exhaustive and evaluative versus supervised) in machine learning and examples of learning problems (bandits, tabular reinforcement learning, reinforcement learning, contextual bandits and supervised machine learning) that result from their combination.

an inherent exploration–exploitation trade-off as the learner is trying to perform well, but also needs to test out courses of action that could possibly be better than the current best guess.

With increasing attention being paid to personalized medicine[4], bandit algorithms could become popular again in the health-care setting. However, the main driver of recent interest in these algorithms is the problem of online content delivery. A company providing web pages to a large user base has many decisions it can make concerning the organization of the information it is providing. Should the page include a large image or a smaller image with space for a brief textual description? A simple formulation of this decision is as a two-armed bandit problem. Each time the system needs to present a page to the user, it chooses between layout A and layout B. It can then estimate the appropriateness of its decision by whether the user clicks on the presented link and, if so, how long the user spends reading and exploring the resulting pages. This information can then be used to inform future layout decisions.

Good algorithms for this problem have been known for more than 50 years. However, increased attention recently has resulted in new analyses and a deeper understanding of which algorithms are most effective in which situations. One popular approach is the upper confidence bound (UCB) algorithm[5]. The idea of the algorithm is quite simple — on the basis of the number of times each alternative has been tested and the outcomes of these tests, the algorithm estimates the likelihood that a link will be selected (its 'click-through rate') for layout A and B separately. Statistical confidence bounds of the form 'with 95% probability, A has a click-through rate of 5%, plus or minus 10%' are computed. The upper confidence bound is the sum of the estimated click-through rate and the confidence bound; 15%, in this case. There are two reasons a condition might have a higher upper confidence bound: it has a higher estimated click-through rate (because it seems to be a good option — exploitation is warranted) or it has a wider confidence range (because it has not received as many tests — exploration is warranted). The UCB heuristic, then, is to always choose the option with the highest upper confidence bound resulting in either high reward or valuable data for learning. UCB is related to an earlier algorithm called interval estimation[6], but it is defined so it achieves excellent guarantees on its long-run regret — roughly speaking, it minimizes the number of clicks lost owing to showing the wrong version of the page.

The UCB algorithm, as described, pays no attention to context. Its choices compare well with the best selection averaged over the entire population of users. Modern web pages have additional information that can be brought to bear, however. The situation at display time, $x$, includes facts that the system has about the user — recent pages visited, possibly demographic information and perhaps even a record of the user's recent purchases. The system can use this information to create the most valuable, interesting or engaging page possible. For example, a news site might select a business or sports headline tailored to the current user's preferences. We, thus, need a way to make good decisions in spite of both evaluative feedback and sampled feedback, a setting that combines aspects of $k$-armed bandits and supervised learning, now known as contextual bandits. UCB has been extended for use in this context[7]. Indeed, contextual bandit algorithms are increasingly being used by companies with a major Web presence.

A modern approach to bandit problems with quite deep roots is a method now known as Thompson sampling[8]. It bears some similarity to a phenomenon known in psychology as probability matching[9]; as a bandit algorithm, the idea is to use the observations obtained thus far to maintain a posterior distribution over the click-through rate of the alternatives. When it is time to make a decision, the algorithm samples a click-through rate for each alternative from its corresponding posterior distribution, then selects the alternative that was assigned the highest value. Or, more simply, it chooses among the alternatives proportionally to the probability that each is best. Thompson sampling behaves somewhat like a noisier version of UCB, in that uncertain alternatives or confident-but-high-scoring alternatives are most likely to be selected. It can be used in similar scenarios as UCB and can also be shown to possess similar guarantees on its performance[10,11]. One highly desirable property, however, is that it provides a direct way of applying the emerging class of non-parametric Bayesian methods[12] to decision making — if you can model a process probabilistically, you can use that model to make a selection. This makes it very well suited to decision making in contextual bandit problems.

## Temporal difference learning

Learning from sequential feedback, also known as the temporal credit assignment problem, is an essential challenge faced by decision makers whose choices have both immediate and indirect effects. The essence of the idea can be illustrated through noughts and crosses. Imagine the computer is playing a nought against a strong opponent playing a cross. In a series of boards (Fig. 2), nought makes two moves (B and D) and ultimately loses. Which move should be blamed for the loss? A natural assumption is that both moves participated in the loss and therefore both are equally responsible, or that the move that was closer in time to the loss (D) has more responsibility. In this case, however, it was one of nought's earlier moves (B) that set the stage for its defeat.

The concept of temporal difference learning[13] provides an algorithmic way to make predictions about the long-term implications of selections. Over a series of games, the learner can estimate whether she is more likely to win or lose from each of the boards it encounters. We can write $V(A) = 0$ to represent the fact that nought is likely to tie from board A, and $V(B) = V(C) = V(D) = V(E) = -1$ to represent the fact that nought should be expected to lose from the other boards. The temporal difference error between two consecutive boards $x$ and $x'$ is $V(x') - V(x)$. Here, the temporal difference error is 0 everywhere except for the transition from A to B — it is the decision that changed nought's situation from a tie to a loss that should be held responsible for the loss.

Leveraging its utility as a marker of successful and unsuccessful decisions, the temporal difference error can be used as a learning signal for improving the value estimates. In particular, an ideal predictor of value will have the property that the long-term prediction for a situation $x$ should be the same as the expected immediate outcome plus the prediction for the resulting situation $x'$: $V(x) \approx E_{x'}[r(x') + V(x')]$. Here, $r(x')$

represents the feedback received from the environment for the transition to $x'$. Classic temporal difference methods set about minimizing the difference between these quantities.

Reinforcement learning in the face of evaluative, sampled and sequential feedback requires combining methods for temporal credit assignment with methods for generalization. Concretely, if the $V$ values are represented with a neural network or similar representation, finding predictions such that $V(x) \approx Ex'[r(x') + V(x')]$ can lead to instability and divergence in the learning process[14,15].

Recent work[16] has modified the goal of learning slightly by incorporating the generalization process into the learning objective itself. Specifically, consider the goal of seeking $V$ values such that $V(x) \approx \prod E_{x'}[r(x') + V(x')]$ where $\prod$ is the projection of the values resulting from their representation by the generalization method. When feedback is exhaustive and no generalization is used, $\prod$ is the identity function and this goal is no different from classic temporal difference learning. However, when linear functions are used to represent values, this modified goal can be used to create provably convergent learning algorithms[17].

This insight was rapidly generalized to non-linear function approximators that are smooth (locally linear)[18], to the control setting[19], and to the use of eligibility traces in the learning process[20]. This line of work holds a great deal of promise for creating reliable methods for learning effective behaviour from very weak feedback.

## Planning

A closely related problem to that of temporal credit assignment is planning. The essence of both of these problems is that when actions have both immediate and situation-altering effects, decisions need to be sensitive to both immediate and long-term outcomes. Temporal difference learning provides one mechanism for factoring long-term outcomes into the decision process. Planning is another in that it considers the implications of taking entire sequences of actions and explicitly makes a trade-off among them. Planning has long been studied in artificial intelligence[21], but research in reinforcement learning has brought about a number of powerful innovations.

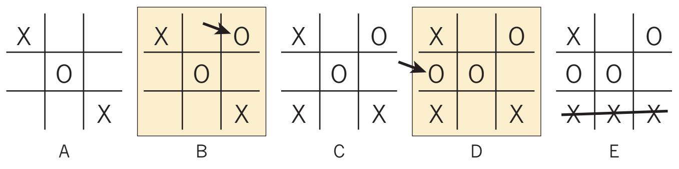


**Figure 2 | A series of boards in noughts and crosses leading to a loss for nought.** Against a strong opponent, cross (moves A, C and E), nought makes two moves (B and D) and ultimately loses. In this example, nought's earlier move (B) set the stage for the defeat.

A particularly challenging kind of planning that is relevant in the reinforcement-learning setting is planning under uncertainty. Here, outcomes of decisions are modelled as being drawn from a probability distribution over alternatives conditioned on the learner's decision (Box 1). A familiar example of this kind of planning is move selection in board games. If a computer program is playing against a human, it needs to make moves that are likely to lead to a win despite not knowing precisely which moves its opponent will take. For a game such as noughts and crosses, a computer program can exhaustively enumerate all the reachable boards — the so-called game tree — and calculate

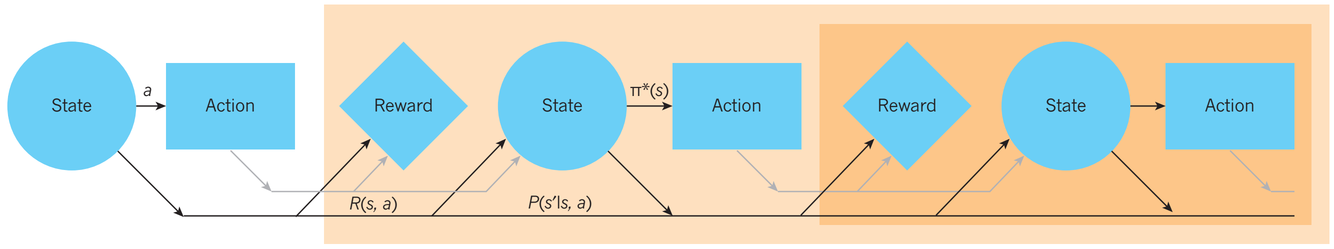
## BOX 1

## Markov decision processes specify setting and tasks

Reinforcement learning is characterized as an interaction between a learner and an environment that provides evaluative feedback. The environment is often conceptualized as a Markov decision process — a formal mathematical model that dates back to the 1950s (refs 55, 56). A Markov decision process is defined by a set of states $S$ (situations in which a decision can be made), and actions $A$ (the decisions or interventions the decision maker can select). These quantities can be taken to be finite, but continuous state and action spaces are often valuable for capturing interactions in important reinforcement-learning applications such as robotic control[57]. A transition function $P(s'|s,a)$ defines the probability of the state changing from $s$ to $s'$ under the influence of action $a$. It specifies the 'physics' or 'dynamics' of the environment.

The decision maker's task is defined by a reward function $R(s,a)$ and discount factor $\gamma \in [0, 1]$. Rewards are delivered to the decision maker with each transition and maximizing the cumulative discounted expected reward is its objective. Specifically, the decision maker seeks a behaviour $\pi^*$ mapping states to actions generating a sequence of rewards $r_0, r_1, r_2, r_3, \ldots$ such that $E_{r0,\, r1,\, \ldots}[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots]$ is as large as possible. The relation between the environmental interaction (state, action, reward, state, action, reward, …) and the cumulative discounted expected reward is captured by the Bellman equation (Figure) for the optimal state–action value function $Q^*$. The solution to the Bellman equation can be used to define optimal behaviour by $\pi^*(s) = \arg\max_a Q^*(s, a)$. The cumulative discounted expected reward for the policy that takes action $a$ from state $s$ and then behaving optimally thereafter is the immediate reward received plus the discounted expected value of the cumulative discounted expected reward from the resulting state $s'$ given that the best action is chosen.

Planning methods use knowledge of $P$ and $R$ to compute a good policy $\pi$. Reinforcement-learning methods have access to $P$ and $R$ only through their ongoing interactions with the environment and must learn good behaviour. Note that the Markov decision process model captures both sequential feedback and the more specific one-shot feedback (when $P(s'|s, a)$ is independent of $s$ and $a$). It also captures both exhaustive feedback and the more general sampled feedback (when states are represented by features that allow $R(s, a)$ to be represented compactly).



$$Q^*(s, a) = R(s, a) + \gamma \Sigma P(s'|s, a) \max_{a'} Q^*(s', a')$$

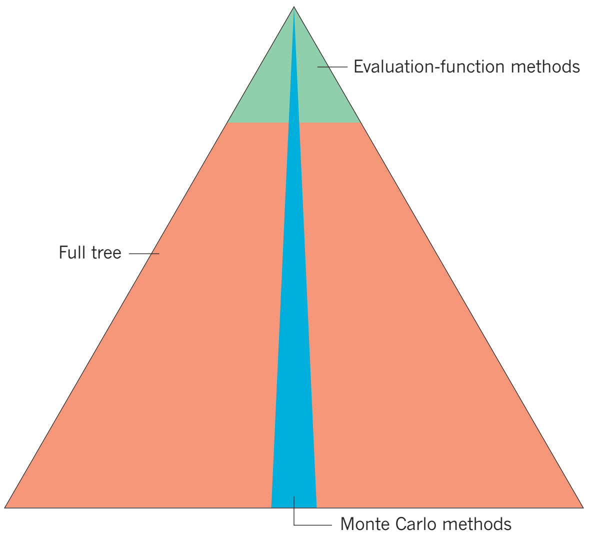


**Figure 3 | Schematic comparison of evaluation-function and Monte Carlo methods for planning.** The top of the tree represents the current state for which a decision is needed. The red area represents the entire search tree — too large to be examined. The green area represents the nodes of the tree visited by evaluation-function methods — they expand to a fixed depth and then use an evaluation function to approximate the value received at the end of the game. The blue area represents the nodes of the tree visited by Monte Carlo methods — they consider a subset of nodes that ignore some actions (making it narrow) but continue to the leaves of the search tree (making it deep). Some decision problems are much more amenable to this type of approximation.

whether its moves will lead to a win, a loss or a tie from each. In a game such as chess, however, the size of the game tree is astronomical and the best an algorithm can hope for is to assemble a representative sample of boards to consider. The classic approach for dealing with this problem is to use an evaluation function. Programs search as deeply in the game tree as they can, but if they cannot reach the end of the game where the outcome is known, a heuristic evaluation function is used to estimate how the game will end. Even a moderately good guess can lead to excellent game play in this context — evaluation-function methods have been used in chess from the earliest days of artificial intelligence to the decision making in the Deep Blue system that beat the World Chess Champion Garry Kasparov[22].

A powerful variation of this idea is to use machine learning to improve the evaluation function over time. Here, temporal difference learning can be used in simulated games to provide opportunities to learn an accurate evaluation function represented as, for example, a neural network. Celebrated examples of the use of temporal difference methods combined with generalization were in the domains of checkers[23] and backgammon[24]. A similar approach was responsible for Daily Double wagering in the Watson system for playing the US television game show Jeopardy![25].

In spite of their impressive, indeed superhuman, performance in some games, evaluation-function methods have not been universally successful. A notable example is the game of Go, in which good evaluation functions have been hard to come by, resulting in subpar game play. A relatively recent breakthrough was the introduction of Monte Carlo tree search (MCTS) methods that swap an estimate of the evaluation function for an estimate of a good policy. Whereas evaluation-function methods try all possible actions out to a fixed depth and estimate the value of the resulting boards, Monte Carlo methods search all the way to the end of the game where evaluations are exact, but use a heuristic rule to suggest which subset of actions are worth considering (Fig. 3). Use of the upper confidence intervals in trees (UCT) algorithm[26] resulted in a quantum leap in performance in Go[27]. It leverages the UCB algorithm to intelligently allocate search resources at each point in the tree.

## Model-based reinforcement learning

To plan, a learning algorithm needs to be able to predict the immediate outcomes of its actions. Since it can try an action and observe the effects right away, this part of the problem can be addressed using supervised learning methods. In model-based reinforcement-learning approaches, experiences are used to estimate a transition model, mapping from situations and decisions to resulting situations, and then a planning approach is used to make decisions that maximize predicted long-term outcomes.

Model-based reinforcement-learning methods and model-free methods such as temporal differences strike a different balance between computational expense and experience efficiency — to a first approximation, model-free methods are inexpensive, but slow to learn; whereas model-based methods are computationally intensive, but squeeze the most out of each experience (Box 2). In applications such as learning to manage memory resources in a central processing unit, decisions need to be made at the speed of a computer's clock and data are abundant, making temporal difference learning an excellent fit[28]. In applications such as a robot helicopter learning acrobatic tricks, experience is slow to collect and there is ample time offline to compute decision policies, making a model-based approach appropriate[29]. There are also methods that draw on both styles of learning[30] to try to get the complementary advantages.

Model-based algorithms can make use of insights from supervised learning to deal with sampled feedback when learning their transition

**BOX 2**

## Two approaches to tabular reinforcement learning

Tabular reinforcement-learning problems are those in which the state–action space can be explored exhaustively. Although challenging, there are a number of relatively simple algorithms that can be used effectively in this setting. Q-learning[58] is a model-free algorithm that uses transition experience of the form $<s, a, r, s'>$ (from state $s$, action $a$ resulted in reward $r$ and next state $s'$) to improve an estimate $\hat{Q}$ of the optimal state–action value function $Q^*$. In particular, since $r+\gamma \max_{a'} Q^*(s', a')$ is an unbiased estimator of $Q^*(s, a)$ as defined by the Bellman equation and $\hat{Q}$ is an estimate of $Q^*$, the estimate can be updated as:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)),$$

where $\alpha$ is a learning-rate parameter that controls how new estimates are iteratively blended together over time. Indeed, if the learning rate is decayed at an appropriate rate,

$$\hat{Q}(s,a) \rightarrow Q^*(s,a)$$

for all state–action pairs[59].

A simple model-based approach to tabular reinforcement learning uses the observed transition experience to estimate $R(s, a)$ (by averaging all rewards received from state $s$ and action $a$) and $P(s'|s, a)$ (by the fraction of transitions from state $s$ and action $a$ that transition to state $s'$). Solving the Bellman equation — through planning algorithms such as value iteration, policy iteration, UCT (upper confidence intervals in trees) or linear programming methods — using these estimates leads to an approximation $\hat{Q}$ of the state–action value function that converges to $Q^*$ in the limit of infinite data.

**BOX 3**

## Self-aware learning in reinforcement learning

The KWIK ('knows what it knows') learning setting[33] provides a powerful set of algorithms for rapidly learning accurate predictions and confidently distinguishing learned from not-yet-learned instances. In the context of reinforcement-learning algorithms, this ability to tag predictions as uncertain can be exploited to drive exploration by substituting optimistic estimates for those that remain unknown.

A growing set of KWIK-learnable function classes has been identified. As a simple example, consider trying to learn a function of the form $f(x)$ = is $x$ divisible by $k$? for some unknown natural number $k$. The Table shows what a KWIK learning algorithm for this problem would predict given training examples $f(6)$ = yes and $f(20)$ = no. It would know $f(6)$ = yes and $f(20)$ = no from these examples. It would be able to reason that the unknown $k \in \{2, 3, 6\}$ from these examples and therefore that $f(25)$ = no. By contrast, $f(15)$ = I don't know because it would be 'yes' if $k = 3$ and 'no' if $k = 6$. Despite this lingering ambiguity, it could conclude that $f(24)$ = yes, since that would be the answer for any of the remaining possible values of $k$. This last element lies at the heart of all KWIK-learning algorithms — not 'wasting' its I don't know answers on examples whose outputs can be inferred from the data.

For noisy function classes, a KWIK learner must be able to predict the probability of various outputs given an input. This property is important when learning Markov decision process transition functions. Some challenging function classes for which KWIK learning algorithms are known include noisy linear regression, and noisy union. Noisy union, also called the 'adaptive $k$ meteorologist' problem[60], lets the learner choose a function from a finite class when there is a non-zero probability that training examples are corrupted.

**Table 1 | A training set ($f(6)$ = yes, $f(20)$ = no) for a function of the form '$f(x)$ = is $x$ divisible by $k$?' for some unknown natural number $k$, and the predictions made by a valid KWIK algorithm.**

| Input $x$ | Output $f(x)$ | Prediction for $f(x)$ |
|---|---|---|
| 6 | Yes | Yes |
| 15 | | I don't know |
| 20 | No | No |
| 24 | | Yes |
| 25 | | No |

models and can tackle the problem of sequential feedback by planning actions to achieve their goals. The issue of evaluative feedback — the fact that the system can only learn about transitions it has actually experienced — makes it very important that learners balance exploration and exploitation. Several methods are known for achieving guaranteed efficiency in model-based reinforcement learning[31,32] by explicitly modelling the certainty of predictions and, much like UCB, taking an optimistic view of the uncertainty. Concretely representing the uncertainty in learning leads to the framework 'knows what it knows'[33] (Box 3), which has been shown to work well with planning algorithms in model-based reinforcement-learning settings.

## Empirical methods

An important turning point in the development of practical supervised learning methods was a shift from artificial learning problems to learning problems grounded in measured data and careful experimental design[34]. The reinforcement-learning community is showing signs of a similar evolution, which promises to help to solidify the technical gains described in the previous sections. Of particular interest are new evaluation methodologies that go beyond case studies of individual algorithm learning in individual environments to controlled studies of different learning algorithms and their performance across multiple natural domains. In supervised learning, data-set collections such as the University of California, Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/) and the Weka Machine Learning library (http://www.cs.waikato.ac.nz/ml/weka/) facilitate this kind of research by providing data for learning problems and a suite of algorithms in a modular form. A challenge for the study of reinforcement learning, however, is that learning takes place through active exploration between the learner and her environment. Environments are almost always represented by simulators, which can be difficult to standardize and share, and are rarely realistic stand-ins for the actual environment of interest.

One noteworthy exception is the Arcade Learning Environment[35], which provides a reinforcement-leaning-friendly interface for an eclectic collection of more than 50 Atari 2600 video games. Although video games are by no means the only domain of interest, the diversity of game types and the unity of the interface makes the Arcade Learning Environment an exciting platform for the empirical study of machine reinforcement learning. An extremely impressive example of research enabled by this repository is a recent study of deep learning in the reinforcement-learning setting. Researchers combined a set of existing and novel algorithm elements to create an agent capable of 'human-level' performance[36] across the set of games. It remains to be seen whether the algorithmic approaches highlighted in this work become standard weapons in the reinforcement-learning arsenal, but the remarkable success on these challenging learning problems has attracted a great deal of attention and is likely to generate advances in the theory and practice of reinforcement learning.

## A perspective on future impacts

There are several current directions that promise considerable impacts in the years to come.

### Offline evaluation

One challenge to designing practical, reliable learning systems for real-life reinforcement-learning problems is that the learning problem is carried out online. That is, in contrast to supervised learning systems that work with a training set that contains measured, but frozen, data, reinforcement learning is defined with regard to a continual interaction between the learner and an environment.

To overcome this difficulty, several efforts are underway to collect static data sets in a way that allows them to be used for evaluating reinforcement-learning algorithms. In the biostatistics community, a new type of clinical trial[37] called a sequential, multiple assignment randomized trial (SMART) has been developed that makes it possible to gather treatment responses in a way that supports the synthesis of adaptive treatment plans. That is, the trial assesses the effectiveness of treatments expressed as flow charts in which an intervention is applied and, depending on the patient's response, the plan follows up with different possible interventions. A typical SMART can evaluate a flow chart containing eight different possible treatment plans using half the participants that would otherwise be needed.

SMARTs evaluate flow-chart-like treatment plans, but not complete learning algorithms. Learning algorithms can express extremely complex contingent relationships between the data acquired during learning and the decisions made in response to it. Consider a simple two-armed bandit problem over a set of ten steps. The tenth decision in the sequence is conditioned on the outcomes of all the previous decisions. Thus, for each of the $2^9 = 512$ possible contexts, our bandit algorithm needs to make a good choice. Longer learning periods and contextual decisions make the number of possibilities to consider astronomically high. Despite this daunting combinatorial explosion, offline evaluation procedures for contextual bandit problems have been proposed[38]. The key idea is to collect a static (but enormous) set of contextual bandit decisions by running a uniform random selection algorithm on a real problem (a news article recommendation, for example). Then, to test a bandit algorithm using this collection, present the algorithm with the same series of contexts. Whenever the selection made by the algorithm does not match the selection made during data collection, the algorithm is forced to forget what it saw. Thus, as far as the algorithm 'knows', all of its choices during learning matched the choices made during data collection. The end result is an unbiased evaluation of a dynamic bandit algorithm using static pre-recorded data.

One reason that this approach works for contextual bandit problems is that — apart from the state of the learning algorithm — there are no dependencies from one round of decision making to the next. So, skipping many rounds because the algorithm and the data collection do not match does not change the fundamental character of the learning problem. In the presence of sequential feedback, however, this trick no longer applies. Proposals have been made for creating reinforcement-learning data repositories[39] and for evaluating policies using observational data sets[40], but no solid solution for evaluating general learning algorithms has yet emerged.

## Finding appropriate reward functions

Reinforcement-learning systems strive to identify behaviour that maximizes expected total reward. There is a sense, therefore, that they are 'programmable' through their reward functions — the mappings from situation to score. A helpful analogy to consider is between learning and program interpretation in digital computers: a program (reward function) directs an interpreter (learning algorithm) to process inputs (environments) into desirable outputs (behaviour). The vast majority of research in reinforcement learning has been into the development of effective interpreters with little attention paid to how we should be programming them (providing reward functions).

There are a few approaches that have been explored for producing reward functions that induce a target behaviour given some conception of that behaviour. As a simple example, if one reward function for the target behaviour is known, the space of behaviour-preserving transformations to this reward function is well understood[41]; other, essentially equivalent, reward functions can be created. If a human teacher is available who can provide evaluative feedback, modifications to intended target reinforcement-learning algorithms can be used to search for the target behaviour[42–44]. The problem of inverse reinforcement learning[45,46] addresses the challenge of creating appropriate reward functions given the availability of behavioural traces from an expert executing the target behaviour. It is called 'inverse' reinforcement learning because the learner is given behaviour and needs to generate a reward function instead of the other way around. These methods infer a reward function that the demonstrated behaviour optimizes. One can turn to evolutionary optimization[47] to generate reward functions if there is an effective way to evaluate the appropriateness of the resulting behaviour. One advantage of the evolutionary approach is that, among the many possible reward functions that generate good behaviour, it can identify ones that provide helpful but not too distracting hints that can speed up the learning process.

Looking forward, new techniques for specifying complex behaviour and translations of these specifications into appropriate reward functions are essential. Existing reward-function specifications lack the fundamental ideas that enable the design of today's massive software systems, such as abstraction, modularity and encapsulation. Analogues of these ideas could greatly extend the practical utility of reinforcement-learning systems.

## Reinforcement learning as a cognitive model

The term reinforcement learning itself originated in the animal-learning community. It had long been observed that certain stimuli, if present after an animal selects a particular behavioural action, cause that behavioural action to become more frequent in the future. These stimuli became known as reinforcers because of their role in strengthening behaviour. The pioneers of artificial intelligence were aware of this idea[48,49] and used the term to describe computational processes that would similarly increase the intensity or likelihood of a response. Indeed, according to the software tool Google Ngrams (which charts the frequency of phrases in its database of published books; https://books.google.com/ngrams), the appearance of 'reinforcement learning' reached a local maximum in the mid-1960s, as it was in use by researchers studying the theory of animal learning, education and cybernetics.

The term lost ground through the 1970s and early 1980s, but was picked up again by computer scientists in the mid-to-late 1980s[13,50] and became a central area of study in machine learning. A subtle but important difference was introduced during this second wave. The term 'reinforcement' came to refer to a generic reward signal that needs to be predicted over time instead of the activity of strengthening a behavioural response. Classic learning problems could still be discussed in this way, but the door opened to a wide variety of other powerful algorithmic approaches, such as temporal difference learning, for which behavioural strengthening was not a direct design goal.

As already discussed, the predictive view proved to be powerful and productive. Recent work in neuroscience and cognitive psychology has leveraged this revised understanding of reinforcement learning to create new insights into biological learning. Of particular note is the discovery of a temporal difference signal encoded in the firing of dopamine neurons[51], which has created a bridge between computer scientists and cognitive neuroscientists centred on reinforcement-learning algorithms[52,53]. In addition to temporal difference methods, cognitive scientists have found the distinction between model-based (goal-directed) and model-free (habitual) learning to be a fertile one. For example, some aspects of how people make moral judgments[54] can be fruitfully described by associating action-based preferences (do not do A because it is wrong) with model-free learning and outcome-based preferences (do not do A because it will lead to a bad situation) with model-based learning.

In October 2013, the first Multi-disciplinary Conference on Reinforcement Learning and Decision Making was held at Princeton University in New Jersey. In addition to computer scientists, psychologists and neuroscientists, the invited speakers included engineers, roboticists, a zoologist, a psychiatrist and a behavioural game theorist. The talks were notable both for the breadth of topics and the convergence of the underlying conceptual framework discussed in this Review. As our formal and practical understanding of learning from evaluative feedback expands, so does the community of scientists and engineers who can contribute to and benefit from these ideas. ■

**Received 11 January; accepted 28 April 2015.**

1. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (MIT Press, 1998).
**This book is the definitive reference on computational reinforcement learning.**
2. Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: a survey. *J. Artif. Intell. Res.* **4,** 237–285 (1996).
3. Berry, D. A. & Fristedt, B. *Bandit Problems: Sequential Allocation of Experiments* (Chapman and Hall, 1985).
4. Shrager, J. & Tenenbaum, J. M. Rapid learning for precision oncology. *Nature Rev. Clin. Onco.* **11,** 109–118 (2014).
5. Auer, P., Cesa-Bianchi, N. & Fischer, P. Finite-time analysis of the multi-armed bandit problem. *Mach. Learn.* **47,** 235–256 (2002).
6. Kaelbling, L. P. *Learning in Embedded Systems* (MIT Press, 1993).
7. Li, L., Chu, W., Langford, J. & Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proc. 19th International World Wide Web Conference* 661–670 (2010).
8. Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **25,** 285–294 (1933).
9. West, R. F. & Stanovich, K. E. Is probability matching smart? Associations between probabilistic choices and cognitive ability. *Mem. Cognit.* **31,** 243–251 (2003).
10. May, B. C., Korda, N., Lee, A. & Leslie, D. S. Optimistic Bayesian sampling in contextual-bandit problems. *J. Mach. Learn. Res.* **13,** 2069–2106 (2012).

11. Bubeck, S. & Liu, C.-Y. Prior-free and prior-dependent regret bounds for Thompson sampling. In *Proc. Advances in Neural Information Processing Systems* 638–646 (2013).
12. Gershman, S. & Blei, D. A tutorial on Bayesian nonparametric models. *J. Math. Psychol*. **56,** 1–12 (2012).
13. Sutton, R. S. Learning to predict by the method of temporal differences. *Mach. Learn.* **3,** 9–44 (1988).
14. Boyan, J. A. & Moore, A. W. Generalization in reinforcement learning: safely approximating the value function. In *Proc. Advances in Neural Information Processing Systems* 369– 376 (1995).
15. Baird, L. Residual algorithms: reinforcement learning with function approximation. In *Proc. 12th International Conference on Machine Learning* (eds Prieditis, A. & Russell, S.) 30–37 (Morgan Kaufmann, 1995).
16. Sutton, R. S. *et al*. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proc. 26th Annual International Conference on Machine Learning* 993–1000 (2009).
17. Sutton, R. S., Maei, H. R. & Szepesvári, C. A convergent O(n) temporal-difference algorithm for off-policy learning with linear function approximation. In *Proc. Advances in Neural Information Processing Systems* 1609–1616 (2009).
18. Maei, H. R. *et al*. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Proc. Advances in Neural Information Processing Systems* 1204–1212 (2009).
19. Maei, H. R., Szepesvári, C., Bhatnagar, S. & Sutton, R. S. Toward off-policy learning control with function approximation. In *Proc. 27th International Conference on Machine Learning* 719–726 (2010).
20. van Hasselt, H., Mahmood, A. R. & Sutton, R. S. Off-policy TD(λ) with a true online equivalence. In *Proc. 30th Conference on Uncertainty in Artificial Intelligence* 324 (2014).
21. Russell, S. J. & Norvig, P. *Artificial Intelligence: A Modern Approach* (Prentice–Hall, 1994).
22. Campbell, M., Hoane, A. J. & Hsu, F. H. Deep blue. *Artif. Intell*. **134,** 57–83 (2002).
23. Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. Develop*. **3,** 211–229 (1959).
24. Tesauro, G. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput*. **6,** 215–219 (1994).
**This article describes the first reinforcement-learning system to solve a truly non-trivial task.**
25. Tesauro, G., Gondek, D., Lenchner, J., Fan, J. & Prager, J. M. Simulation, learning, and optimization techniques in Watson's game strategies. *IBM J. Res. Develop.* **56,** 1–11 (2012).
26. Kocsis, L. & Szepesvári, C. Bandit based Monte-Carlo planning. In *Proc. 17th European Conference on Machine Learning* 282–293 (2006).
**This article introduces UCT, the decision-making algorithm that revolutionized gameplay in Go.**
27. Gelly, S. *et al*. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM* **55,** 106–113 (2012).
28. İpek. E., Mutlu, O., Martínez, J. F. & Caruana, R. Self-optimizing memory controllers: a reinforcement learning approach. In *Proc. 35th International Symposium on Computer Architecture* 39–50 (2008).
29. Ng, A. Y., Kim, H. J., Jordan, M. I. & Sastry, S. Autonomous helicopter flight via reinforcement learning. In *Proc. Advances in Neural Information Processing Systems* http://papers.nips.cc/paper/2455-autonomous-helicopter-flight-via-reinforcement-learning (2003).
30. Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. 7th International Conference on Machine Learning* 216–224 (Morgan Kaufmann, 1990).
31. Kearns, M. J. & Singh, S. P. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.* **49,** 209–232 (2002).
**This article provides the first algorithm and analysis that shows that reinforcement-learning tasks can be solved approximately optimally with a relatively small amount of experience.**
32. Brafman, R. I. & Tennenholtz, M. R-MAX — a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res*. **3,** 213–231 (2002).
33. Li, L., Littman, M. L., Walsh, T. J. & Strehl, A. L. Knows what it knows: a framework for self-aware learning. *Mach. Learn.* **82,** 399–443 (2011).
34. Langley, P. Machine learning as an experimental science. *Mach. Learn.* **3,** 5–8 (1988).
35. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The arcade learning environment: an evaluation platform for general agents. *J. Artif. Intell. Res*. **47,** 253–279 (2013).
36. Mnih, V. *et al*. Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
**This article describes the application of deep learning in a reinforcement-learning setting to address the challenging task of decision making in an arcade environment.**
37. Murphy, S. A. An experimental design for the development of adaptive treatment strategies. *Stat. Med*. **24,** 1455–1481 (2005).
38. Li, L., Chu, W., Langford, J. & Wang, X. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proc. 4th ACM International Conference on Web Search and Data Mining* 297–306 (2011).
39. Nouri, A. *et al*. A novel benchmark methodology and data repository for real-life reinforcement learning. In *Proc. Multidisciplinary Symposium on Reinforcement Learning*, Poster (2009).
40. Marivate, V. N., Chemali, J., Littman, M. & Brunskill, E. Discovering multi-modal characteristics in observational clinical data. In *Proc. Machine Learning for Clinical Data Analysis and Healthcare NIPS Workshop* http://paul.rutgers.edu/~vukosi/papers/nips2013workshop.pdf (2013).
41. Ng, A. Y., Harada, D. & Russell, S. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conference on Machine Learning* 278–287 (1999).
42. Thomaz, A. L. & Breazeal, C. Teachable robots: understanding human teaching behaviour to build more effective robot learners. *Artif. Intell*. **172,** 716–737 (2008).
43. Knox, W. B. & Stone, P. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proc. 5th International Conference on Knowledge Capture* 9–16 (2009).
44. Loftin, R. *et al*. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proc. 28th Association for the Advancement of Artificial Intelligence Conference* https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8579 (2014).
45. Ng, A. Y. & Russell, S. Algorithms for inverse reinforcement learning. In *Proc. International Conference on Machine Learning* 663–670 (2000).
46. Babes, M., Marivate, V. N., Littman, M. L. & Subramanian, K. Apprenticeship learning about multiple intentions. In *Proc. International Conference on Machine Learning* 897–904 (2011).
47. Singh,S., Lewis, R.L., Barto, A.G. & Sorg, J. Intrinsically motivated reinforcement learning: an evolutionary perspective. *IEEE Trans. Auto. Mental Dev.* **2,** 70–82 (2010).
48. Newell, A. The chess machine: an example of dealing with a complex task by adaptation. In *Proc. Western Joint Computer Conference* 101–108 (1955).
49. Minsky, M. L. Some methods of artificial intelligence and heuristic programming. In *Proc. Symposium on the Mechanization of Thought Processes* 24–27 (1958).
50. Sutton, R. S. & Barto, A. G. Toward a modern theory of adaptive networks: expectation and prediction. *Psychol. Rev*. **88,** 135–170 (1981).
51. Schultz, W., Dayan, P. & Montague, P. R. A neural substrate of prediction and reward. *Science* **275,** 1593–1599 (1997).
52. Dayan, P. & Niv, Y. Reinforcement learning and the brain: the good, the bad and the ugly. *Curr. Opin. Neurobiol*. **18,** 185–196 (2008).
53. Niv, Y. Neuroscience: dopamine ramps up. *Nature* **500,** 533–535 (2013).
54. Cushman, F. Action, outcome, and value a dual-system framework for morality. *Pers. Soc. Psychol. Rev*. **17,** 273–292 (2013).
55. Shapley, L. Stochastic games. *Proc. Natl Acad. Sci. USA* **39,** 1095–1100 (1953).
56. Bellman, R. *Dynamic Programming* (Princeton Univ. Press, 1957).
57. Kober, J., Bagnell, J. A. & Peters, J. Reinforcement learning in robotics: a survey. *Int. J. Rob. Res.* **32,** 1238–1274 (2013).
58. Watkins, C. J. C. H. & Dayan, P. Q-learning. *Mach. Learn*. **8,** 279–292 (1992).
**This article introduces the first provably correct approach to reinforcement learning for both prediction and decision making.**
59. Jaakkola, T., Jordan, M. I. & Singh, S. P. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in Neural Information Processing Systems* 6, 703–710 (Morgan Kaufmann, 1994).
60. Diuk, C., Li, L. & Leffner, B. R. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proc. 26th International Conference on Machine Learning* 32–40 (2009).

**Acknowledgements**
The author appreciates his discussions with his colleagues that led to this synthesis of current work.

**Author Information** Reprints and permissions information is available at www.nature.com/reprints. The author declares no competing financial interests. Readers are welcome to comment on the online version of this paper at go.nature.com/cslqwl. Correspondence should be addressed to M.L.L. (mlittman@cs.brown.edu).