From: AAAI Technical Report SS-96-01. Compilation copyright © 1996, AAAI (www.aaai.org). All rights reserved.

A GENERAL METHOD FOR MULTI-AGENT REINFORCEMENT LEARNING IN UNRESTRICTED ENVIRONMENTS

Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, CH-6900 Lugano
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Previous approaches to multi-agent reinforcement learning are either very limited or heuristic by nature. The main reason is: each agent's environment continually changes because the other agents keep changing. Traditional reinforcement learning algorithms cannot properly deal with this. This paper, however, introduces a novel, general, sound method for multiple, reinforcement learning agents living a single life with limited computational resources in an unrestricted environment. The method properly takes into account that whatever some agent learns at some point may affect learning conditions for other agents or for itself at any later point. It is based on an efficient, stack-based backtracking procedure called "environment-independent reinforcement acceleration" (EIRA), which is guaranteed to make each agents learning history a history of performance improvements (long term reinforcement accelerations). The principles have been implemented in an illustrative multi-agent system, where each agent is in fact just a connection in a fully recurrent reinforcement learning neural net.

BASIC IDEAS

I will first focus on a single agent. In the next section we will see that the generalization to the multi-agent case is straightforward.

Single agent scenario. Consider a single learning agent executing a lifelong action sequence in an unknown environment. The environment is unrestricted — in what follows, it won't even be necessary to introduce a formal model of the environment. Different agent actions may require different amounts of execution time (like in scenarios studied in (Russell and Wefald, 1991; Boddy and Dean, 1994)). Occasionally the environment provides real-valued "reinforcement". The sum of *all* reinforcements obtained between "agent birth" (at time 0) and time $t > 0$ is denoted by $R(t)$. Throughout its lifetime, the agent's goal is to maximize $R(T)$, the cumulative reinforcement at (initially unknown) "death" T. There is only one life. Time flows in *one* direction (no resets to zero).

Policy. The agent's current policy is embodied by modifiable parameters of an arbitrary algorithm mapping environmental inputs and internal states to output actions and new internal states. The number of parameters needs not be fixed. In the first part of this paper, it won't be necessary to introduce a formal description of the policy.

Learning by policy modification processes (PMPs). During certain time intervals in agent life, policy parameters are occasionally modified by a sequence of finite "policy modification processes" (PMPs), sometimes also called "learning processes". Different PMPs may take different amounts of time. The i-th PMP in agent life is denoted PMP_i, starts at time $t_1^i > 0$, ends at $t_2^i < T$, $t_2^i > t_1^i$, and computes policy modification $M(i)$. A non-zero time-interval will pass between t_2^i and t_1^{i+1}, the beginning of PMP_{i+1}. While PMP_i is running, the agent's lifelong interaction sequence with the environment may continue, and there may be reinforcement signals, too. In fact, PMP_i may use environmental feedback to compute $M(i)$ — for instance, by executing a known reinforcement learning algorithm. For the moment, however, I do not care for what exactly happens while PMP_i is running: in what follows, I won't need a formal model of the PMP details. The following paragraphs are only concerned with the question: how does $M(i)$ influence the remaining agent life?

The problem. In general environments, events / actions / experiments occurring early in agent life may influence events / actions / experiments at any later time. For instance, the agent's environmental conditions may be affected by learning processes of other agents (this is a major reason why previous approaches to multi-agent learning are heuristic by nature). In particular, PMP_i may affect the environmental conditions for $PMP_k, k > i$. This is something not properly taken into account by existing algorithms for adaptive control and reinforcement learning (e.g., (Kumar and Varaiya, 1986; Barto, 1989; Watkins and Dayan, 1992; Williams, 1992)), and not even by naive, inefficient, but supposedly infallible exhaustive search among all policy candidates, as will be seen next.

What's wrong with exhaustive search? Apart from the fact that exhaustive search is not considered practical even for moderate search spaces, it also suffers from another, more fundamental problem. Let n be the number of the agent's possible policies. For the sake of the argument, suppose that n is small enough to allow for systematic, sequential generate-and-test of all policies within the agent's life time. Suppose that after all policies have been generated (and evaluated for some time), during the remainder of its life, the agent keeps the policy whose test brought about maximal reinforcement during the time it was tested. In the general "on-line" situation considered in this paper, **this may be the wrong thing to do:** for instance, each policy test may change the environment in a way that changes the preconditions for policies considered earlier. Likewise, the quality of a policy may be affected by policy changes of other agents. A policy discarded earlier may suddenly be the "best" policy (but will never be considered again). Similar things can be said about almost every other search algorithm or learning algorithm — exhaustive search is just a convenient, very simple, but representative example.

But then how to measure performance improvements? Obviously, the performance criterion used by naive exhaustive search (and other search and learning algorithms) is not appropriate for the general set-up considered in this paper. We first have to ask: what is a reasonable performance criterion for such general (but typical) situations? More precisely: if we do not make any assumptions about the environment, can we still establish a sensible criterion according to which, at a certain time, (1) the agent's performance throughout its previous life always kept improving or at least never got worse, and which (2) can be *guaranteed* to be achieved by the agent? Indeed, such a criterion can be defined, as will be seen below. First we will need additional concepts.

Reinforcement/time ratios. Suppose PMP_i started execution at time t_1^i and completed itself at time $t_2^i > t_1^i$. For $t \geq t_2^i$ and $t \leq T$, the reinforcement/time ratio $Q(i,t)$ is defined as

$$Q(i,t) = \frac{R(t) - R(t_1^i)}{t - t_1^i}.$$

The computation of reinforcement/time ratios takes into account all computation time, including time required by PMPs.

Currently valid modifications. After t_2^i, PMP_i's effects on the policy will stay in existence until (a) being overwritten by later PMP_k, $k > i$, or until (b) being countermanded by "EIRA", the method to be described below. To define *valid modifications*, only (b) is relevant: after t_2^i, the policy modification $M(i)$ generated by PMP_i, will remain *valid* as long as EIRA (see below) does not reject $M(i)$ (by restoring the old policy right before PMP_i started).

Reinforcement acceleration criterion (RAC). At time t, RAC is satisfied if for each PMP_i that computed a *currently valid* modification $M(i)$ of the agent's policy,

> (a) $Q(i,t) > \frac{R(t)}{t}$, and (b) for all $k < i$, where PMP_k computed a currently valid $M(k)$: $Q(i,t) > Q(k,t)$.

In words: RAC is satisfied if the beginning of each completed PMP that computed a currently valid modification *until now* has been followed by long-term reinforcement acceleration.

Single experiences. Note that at a given time t, there is only *one* observable training example concerning the long term success of PMP_i, namely $Q(i,t)$. Since life is one-way (neither time nor environment are ever reset), our statistics with respect to the long term success of a given, previous PMP will never consist of more than a single experience.

The method to achieve RAC is called "environment-independent reinforcement acceleration" (EIRA). It uses a stack to trace and occasionally invalidate policy modifications:

Environment-independent reinforcement acceleration (EIRA). EIRA uses an initially empty stack to store information about currently valid policy changes. Occasionally, at times called *"checkpoints"*, this information is used to restore previous policies, such that RAC holds. EIRA is based on two complementary kinds of processes:

(1) Pushing. Suppose PMP_i starts at time t_1^i and ends at t_2^i. Let $I(i)$ denote a record consisting of the values t_1^i, $R(t_1^i)$ (which will be needed to compute $Q(i,t)$ values at later times t), plus the information necessary to restore the old policy (before modification by PMP_i). $I(i)$ is pushed onto the stack. By definition, PMP_i is not finished until the pushing process is completed. From now on, the modification $M(i)$ will remain *valid* as long as $I(i)$ will remain on the stack.

(2) Popping. At certain times called *"checkpoints"* (see below), do:

While none of the 3 conditions below holds, pop the topmost $I(.)$-value off the stack, *invalidate* the corresponding modification $M(.)$, and restore the previous policy:

(1) $Q(i,t) > Q(i',t)$, where $i > i'$, $M(i)$ and $M(i')$ are the two most recent currently valid modifications (if existing), and t denotes the current time,

(2) $Q(i,t) > \frac{R(t)}{t}$, where $M(i)$ is the only existing valid modification,

(3) the stack is empty.

For all PMP_i, *checkpoints* occur right before time t_1^i, such that, by definition, t_1^i coincides with the end of the corresponding popping process (additional checkpoints are possible, however). The time eaten up by pushing processes, popping processes, and all other computations is taken into account (for instance, time goes on as popping takes place).

Theorem 1. Whenever popping is finished, EIRA will have achieved RAC. The nature of the (possibly changing) environment does not matter.

Proof sketch. Induction over the stack contents after popping.

What's going on? Essentially, at each checkpoint, EIRA makes the history of the agent's valid policy modifications consistent with RAC: each modification led to further long term reinforcement speed-ups. Then, *by generalizing from a single experience*, the agent keeps all apparently "good" modifications, expecting they will remain "good", until there is evidence to the contrary at some future checkpoint. Note that EIRA does not care for the nature of the PMPs — for all completed PMP_i (based, e.g., on conventional reinforcement learning algorithms), at each "checkpoint", EIRA will get rid of $M(i)$ if t_1^i was not followed by long-term reinforcement speed-up (note that before countermanding $M(i)$, EIRA will already have countermanded all $M(k), k > i$). No modification $M(i)$ is guaranteed to remain valid forever.

Consequences for Multi-Agent Learning

Basic observation. Now consider the case where there are multiple, interacting, EIRA-learning agents. For each agent, the others are part of the changing (possibly complex) environment. But since EIRA is environment-independent, each agent will still be able to satisfy its RAC after each checkpoint. In cases where all agents try to speed up the same reinforcement signals, and where no agent can speed up reinforcement intake by itself, this automatically enforces "learning to cooperate". *This observation already represents the major point of this paper, whose remainder serves illustration purposes only.*

Illustration: multi-agent EIRA for a recurrent reinforcement learning net. To exemplify the insight above, I implemented the following system consisting of multiple agents, where each agent is in fact just a connection in a fully recurrent neural net (a by-product of this research is a general reinforcement learning algorithm for such nets). Each unit of the net has a directed connection to all units, including itself (recurrency allows for storing representations of previous events). The variable activation of the i-th unit is denoted o_i (initialized at time 0 with 0.0). w_{ij} denotes the real-valued, randomly initialized weight (a variable) on the connection (i,j) from unit j to i. *Each connection's current weight represents its current policy.*

Viewing each connection as a separate agent may seem unconventional: standard AI research is used to more complex agents with full-sized knowledge bases etc. For the purposes of this paper, however, this is irrelevant: EIRA does not care for the complexity of the agents.

Cycles. There is a lifelong sequence of *"cycles"*. A cycle involves the following computations: the activations of the network's input units are set by the environment. Each noninput unit i updates its activation as follows: $o_i \leftarrow 1.0$ with probability $f(net_i)$; $o_i \leftarrow 0.0$ otherwise, where $f(x) = \frac{1}{1+e^{-x}}$, and the *net*-values (initially 0) are sequentially updated as follows: $net_i \leftarrow net_i + \sum_j w_{ij} o_j$. Depending on the activation pattern across the output units, an action is chosen and executed. The action may influence the environment, which occasionally may provide (typically delayed) reinforcement.

Weight stacks. Each connection is an agent. The current weight of a connection represents its current policy. For each connection (i,j), there is an initially empty stack. Following the EIRA principle (see above), whenever a weight is modified (see below), the following values are pushed onto (i,j)'s stack: the current time, the total cumulative reinforcement so far, and the weight before the modification.

Weight modification / restauration by EIRA. For each connection, a checkpoint occurs after each cycle. At each checkpoint, the connection follows the EIRA principle, by sequentially popping entries from its stack and restoring the corresponding previous weight values, until its RAC (see above) is satisfied. In between two checkpoints, a small fraction (5 percent) of all weights are replaced by a randomly chosen value in $[-2.0, 2.0]$ (this simple procedure implements the PMP_i above — of course, more sophisticated PMPs would be possible, but this is irrelevant for the purposes of this paper).

Changing environment. Each single connection's environment continually changes, simply because both network activations and all the other connections in its environment keep changing. However, all connections receive the same global reinforcement signal (whenever there is one). Hence, for each connection, the only way to speed up its *local* reinforcement intake is to contribute to speeding up *global* reinforcement intake. If no connection can solve the task by itself, this enforces learning to cooperate.

Illustrative application. The simple system above was applied to a non-Markovian variant of Sutton's (1991) Markovian maze task (unlike with Sutton's original set-up, the system's input is not a unique representation of its current position). To maximize cumulative reinforcement, the system has to find short paths from start to goal. With a preliminary experiment, after slightly more than 10,000 "trials", the average trial length (number of cycles required to move from start to goal) was 89, down from 2660 in the beginning. This corresponds to a speed-up factor of about 30. In the end, no weight stack had more than 7 entries (each followed by faster reinforcement intake than all the previous ones). Details can be found in (Schmidhuber, 1995), where it is also shown **how the policy can learn to define and execute its own PMPs, and learn to set appropriate checkpoints by itself ("meta-learning" and "self-improvement")**. See (Schmidhuber, 1995; Zhao and Schmidhuber, 1996; Wiering and Schmidhuber, 1996) for more recent work along these lines.

It should be emphasized that the preliminary experiment above in no way represents a systematic experimental analysis, which is left for future work. The only purpose of the current section is to illustrate what has been said in the theoretical part of this paper.

Conclusion

It is easy to show that there cannot be an algorithm for general, unknown environments that is guaranteed to continually increase each agent's reinforcement intake per *fixed* time interval. For this reason, the reinforcement acceleration criterion (RAC) relaxes standard measures of performance improvement, by allowing for consideration of *arbitrary, possibly lifelong* time intervals. For each agent, the method presented in this paper is guaranteed to achieve histories of lifelong performance acceleration according to this relaxed criterion. Whenever such a history is established, EIRA makes the most straight-forward generalization step based on the only available training example: until there is evidence to the contrary at a future checkpoint, EIRA assumes that those modifications that kept appearing useful in the past up until now will keep appearing useful.

Standard AI research may be used to more complex agents than those of the illustrative experiment above. The method presented in this paper, however, does not care for the complexity of the agents. EIRA is a general framework — you can plug in your favorite reinforcement learning algorithm A, especially in situations where the applicability of A is questionable because the environment does not satisfy the preconditions that would make A sound.

Acknowledgments

Thanks for valuable discussions to Sepp Hochreiter, Marco Wiering, Rafal Salustowicz, and Jieyu Zhao (supported by SNF grant 21-43'417.95 "Incremental Self-Improvement").

References

Barto, A. G. (1989). Connectionist approaches for control. Technical Report COINS 89-89, University of Massachusetts, Amherst MA 01003.

Boddy, M. and Dean, T. L. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285.

Kumar, P. R. and Varaiya, P. (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice Hall.

Russell, S. and Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, 49:361–395.

Schmidhuber, J. (1995). Environment-independent reinforcement acceleration. Technical Note IDSIA-59-95, IDSIA.

Schmidhuber, J. (1996). A theoretical foundation for multi-agent learning and incremental self-improvement in unrestricted environments. In Yao, X., editor, *Evolutionary Computation: Theory and Applications.* Scientific Publ. Co., Singapore.

Sutton, R. S. (1991). Integrated modeling and control based on reinforcement learning and dynamic programming. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 471–478. San Mateo, CA: Morgan Kaufmann.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Wiering, M. and Schmidhuber, J. (1996). Plugging Levin search into EIRA. Technical Report IDSIA-1-96, IDSIA.

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

Zhao, J. and Schmidhuber, J. (1996). Learning as the cost function changes. Technical Report IDSIA-2-96, IDSIA.