Generalized Domains for Empirical Evaluations in Reinforcement Learning

Shimon Whiteson s.a.whiteson@uva.nl
Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands

Brian Tanner btanner@cs.ualberta.ca
Department of Computing Science, University of Alberta, Edmonton, Canada

Matthew E. Taylor taylorm@usc.edu
Computer Science Department, University of Southern California, Los Angeles, CA

Peter Stone pstone@cs.utexas.edu
Department of Computer Sciences, University of Texas at Austin, Austin, TX

Abstract

Many empirical results in reinforcement learning are based on a very small set of environments. These results often represent the best algorithm parameters that were found after an ad-hoc tuning or fitting process. We argue that presenting tuned scores from a small set of environments leads to method overfitting, wherein results may not generalize to similar environments. To address this problem, we advocate empirical evaluations using generalized domains: parameterized problem generators that explicitly encode variations in the environment to which the learner should be robust. We argue that evaluating across a set of these generated problems offers a more meaningful evaluation of reinforcement learning algorithms.

1. Introduction

The field of reinforcement learning (RL) aims to develop algorithms for solving *sequential decision problems* (SDPs), in which an autonomous *agent* strives to maximize a scalar reward signal by choosing actions in response to observations in a series of timesteps. At each timestep, the *environment* generates observations and a reward in response to the action emitted by the agent. The agent's behavior is determined by its *policy*, a mapping from sequences of observations to actions.

Since many challenging tasks can be framed as RL problems (e.g., robot control, game playing, and system optimization), improving the effectiveness of RL algorithms also advances the progress of artificial intelligence. Advances in the theory of RL have provided several algorithms with both asymptotic and online performance guarantees. However, these guarantees typically apply to very general classes of SDPs (e.g., all finite *Markov decision processes* (MDPs)) and thus the resulting methods do not exploit the structure of any specific problems.

RL has also been successfully applied to very specialized settings, including backgammon (Tesauro, 1994), elevator control (Crites & Barto, 1998), and helicopter control (Ng et al., 2004). However, these applications often require so much domain knowledge that it is difficult to generalize to wider classes of problems. Finding the middle ground, i.e., methods that exploit problem structure but that are robust at least to qualitatively similar tasks, has proven more difficult in practice.

We believe that good empirical methodologies are important for addressing this problem, as history suggests that they can make it easier to devise practical machine learning methods. For example, the ability to share real data sets, as in the Machine Learning Repository at the University of California, Irvine (Asuncion & Newman, 2007), has enabled researchers to directly compare disparate methods on common benchmarks. These comparisons have helped researchers develop increasingly practical methods and thus contributed to supervised learning's indispensable role in a broad range of real-world applications, such as spam filtering (Bratko et al., 2006), fraud detection (Bolton & Hand, 2002), and bioinformatics (Mitra et al., 2008).

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

We believe that the RL community can make a similar transition if we make a commitment to robust empirical evaluation.

A good empirical methodology should be able to determine when a method is not robust, i.e., when its performance is poor outside of the specific conditions and data on which it was trained. Since the learning process can be thought of as fitting free model parameters, this brittleness is often referred to as *overfitting*. The most well-known form is *data overfitting*, in which learning is overfit to a small sample of training data, e.g., in supervised learning. However, overfitting can also occur at a higher level. In particular, in *method overfitting* (Falkenauer, 1998), the learning method itself, rather than the specific function it learns, is overfit to a particular problem, sometimes with the goal of beating benchmark scores in the literature.

In this paper, we claim that method overfitting is a serious concern for empirical studies of RL algorithms. In particular, we argue that some of the distinguishing characteristics of RL make it especially important to devise empirical methodologies that address method overfitting. We also propose a simple solution, an evaluation framework based on *generalized domains*, which was originally developed for use in the 2008 RL Competition. Generalized domains allow experimenters to explicitly encode the scope of variation to which the RL algorithms should be customized. Instead of trying to avoid overfitting, the objective is to fit the generalized domain as well as possible.

2. Method Overfitting in RL

The application of RL methods requires many implementation choices, including exploration strategies, basis-function selection, learning rates, decay schedules, etc. To be precise, we denote each particular set of implementation choices as a separate *algorithm*. Under this definition, even a simple RL method like table-lookup Sarsa(0) (Rummery & Niranjan, 1994) affords a multitude of possible learning algorithms, because there are many possible values for parameters such as the initial value function, the learning rate, and the exploration strategy. Fitting an RL method to a particular task typically involves selecting appropriate values for these parameters.

The fitting process may involve knowledge-engineering strategies like expert interviews, creating simulation models of the environment, evaluating various ideas under controlled conditions, etc. Fitting may be a laborious manual effort, an automated procedure, or both. Regardless, it is likely to be expensive. A manual effort requires human resources while automated fitting requires repeated experimentation, which costs time, money, and productivity, especially if the environment is a physical system or an expensive simulation. Thus, it is clear why method overfitting is undesirable: if the algorithm is too brittle, it will need to be extensively refit for each problem instance, multiplying the costs described above.

While this is a concern for any machine learning method, we believe that method overfitting is particularly damaging in RL because RL methods are typically designed to work well *on-line*. Hence, they strive to maximize the reward they accrue while they are learning, not just to discover a good policy at the end of learning. To perform well on-line, the agent must be able to efficiently explore an unknown environment. However, a method-overfit algorithm can be customized to the environment, reducing or eliminating the need for effective exploration. Such a method will likely perform poorly in other problems, necessitating a refitting process that will be expensive in terms of on-line reward, as it will require experimenting with suboptimal parameter settings.

Hence, while a supervised learning algorithm that is method-overfit may still be useful (e.g., because only CPU cycles are needed to find a good parameter setting), an RL algorithm that is method-overfit has failed in a fundamental way since it cannot perform well on-line. Therefore, we believe that a good empirical methodology for assessing the on-line performance of RL methods must address method overfitting.

Unfortunately, the methodologies currently in common use do not do so. Many empirical evaluations presented in RL measure performance only on a few SDPs. Usually, the authors perform an ad-hoc fitting process and report results for the specialized algorithm that performs best for each individual SDP. This well-intentioned procedure obscures the cost of creating the fitted algorithm and hides its potential brittleness to other qualitatively similar SDPs of interest.

Consider, as an extreme example, the well-known Mountain Car problem (Sutton, 1996), a task for which the optimal policy is already known due to extensive previous research. Using the standard paradigm of testing methods on particular SDPs, one could evaluate an algorithm based on an extreme form of method overfitting: a "learning algorithm" that employs the optimal policy from the first episode and never changes it. No other algorithm could possibly perform better on this task. If the evaluation were valid, then we should conclude the algorithm is an excellent RL method. However, the opposite is obviously true, since the algorithm cannot learn at all and would fail catastrophically on almost any other task. Clearly, this form of evaluation does not capture an aspect of the problem that we know to be important.

3. Generalized Domains for RL

One way to guard against method overfitting is to use an evaluation framework that measures performance, not on a single SDP, but on some class of SDPs of interest. As a result, an RL algorithm will receive a high score only if it is capable of robust learning across that class. Depending on the goals of the researcher, this class may be broad, (e.g., all possible finite MDPs), or narrow, (e.g., helicopter control under specific conditions but with different wind settings). In either case, using a class of SDPs allows researchers to explicitly specify what kind of robustness is expected from the algorithm. Within that class, algorithms must be robust to perform well. Outside of it, method overfitting can occur and is even encouraged if it leads to better performance within the class.

The idea of devising classes of problem instances is not new: the idea has been used successfully in other areas of computer science like combinatorial auctions (Leyton-brown et al., 2000), as well as within RL (Bhatnagar et al., 2009; UMass, 2009; Kalyanakrishnan & Stone, 2009). Here, we propose a simple formalism for this approach called *generalized domains*. A generalized domain $\mathcal{G} = \langle \Theta, \mathcal{P} \rangle$ consists of:

- Θ, a (possibly infinite) set of SDPs. Each element $\theta \in \Theta$ fully specifies the dynamics and reward structure, therefore defining a particular SDP.
- $\mathcal{P}(\Theta)$: A probability distribution over the set of possible SDPs.

In a typical evaluation procedure, an algorithm is evaluated over a set of independent runs. For each run, a particular SDP is sampled from $\mathcal{P}$. Some performance metric, e.g., cumulative reward accrued during the run, is then averaged across all the runs.

Generalized domains were originally designed for the 2008 RL Competition[1], for which the authors were among the organizers. The competition was an international event in which RL researchers compared the performance of their methods on a suite of challenging domains. Generalized domains were used to try to make these comparisons more rigorous and to encourage participants to submit methods that learn on-line and are not merely hard-coded for particular competition events. Generalized domains are also being used in the 2009 RL Competition[2].

To excel in a generalized domain, a learner must be robust to the variation represented by Θ and $\mathcal{P}$. Except in degenerate cases, no fixed policy will perform well across many settings in Θ. Consequently, for strong performance, substantial learning is required to be successful. However, learning algorithms are still subject to method overfitting outside of Θ. Just as a non-learning agent preloaded with a policy customized for Mountain Car may fail on other tasks, a learning agent customized to a generalized version of Mountain Car may fail on other generalized tasks. In other words, the goal is not to cope with an arbitrary, unknown $\mathcal{P}$ but to perform robustly for a given $\mathcal{P}$.

In this respect, the generalized domain framework strikes an intermediate pose between single-task evaluations and more elaborate models of learning in multiple domains (Baxter, 2000; Wilson et al., 2007), which require robustness to an arbitrary $\mathcal{P}$. While such models are of theoretical interest, it is not clear how to turn them into frameworks for empirical evaluations because, in practice, the learner's knowledge of $\mathcal{P}$ cannot be easily controlled. As researchers get more experience with a benchmark, they learn more about $\mathcal{P}$ and can develop algorithms that are customized to it and not robust to other values of $\mathcal{P}$.

By contrast, in generalized domains, the only robustness required is within $\mathcal{P}$, which can be reliably measured since the agent, no matter how customized to $\mathcal{P}$, has no way of predicting which specific SDP it will face in a given run. While method overfitting is not eliminated, it no longer interferes with the validity of the evaluations, since any type of robustness that is deemed important can be ensured by proper design of Θ and $\mathcal{P}$. If algorithms are proposed that approach the ceiling of possible performance on a generalized domain, more challenging generalized domains can easily be devised by enlarging Θ. The strategy of using new and expanded benchmarks as a defense against method overfitting is consistent with similar suggestions that have been made for the UCI database (Gent, 1999).

4. Future Work

The generalized domain framework proposed here provides one simple way to measure a method's robustness on a range of SDPs. However, the task remains to identify suitable generalized domains that encourage robustness with respect to dimensions of particular interest to researchers or that vary substantially in

[1] `http://2008.rl-competition.org`

[2] `http://2009.rl-competition.org`

real-world problems. We believe this effort ought to be a priority of the RL community and we plan to focus on it in future work.

In particular, we are interested in two different strategies for developing RL benchmarks. In the *top-down* approach, the goal is to design general-purpose methods, i.e., off-the-shelf methods that work well on a broad range of problems and provide a useful baseline when considering a new task. Generalized domains can aid this approach by providing a framework for evaluating learning algorithms across disparate tasks.

By contrast, the *bottom-up* approach is motivated by the belief that direct progress towards general algorithms may be difficult in practice and that performance improvements are more easily achieved by identifying important subclasses of problems and developing specialized methods for them. For example, Lane and Smart (2005) write that "a profitable approach for the future is to cleave RL into a number of subdisciplines, each studying important 'special cases.' By doing so, we will be able to take advantage of the properties of these cases in ways that our current (PO)MDP frameworks are unable to." Generalized domains can be used to formalize these sub-classes, which can begin as highly specific tasks and be gradually broadened as much as possible.

Regardless of whether a top-down or bottom-up approach is employed, we believe that generalized domains have a useful role to play in specifying benchmarks, guarding against overfitting, and facilitating the development of increasingly practical methods for reinforcement learning.

References

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, *12*, 149–198.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., & Lee, M. (2009). Natural actor-critic algorithms. *Automatica*. To appear.

Bolton, R., & Hand, D. (2002). Statistical fraud detection: a review. *Statistical Science*, *17*, 235–255.

Bratko, A., Filipic, B., Cormack, G., Lynam, T., & Zupan, B. (2006). Spam filtering using compression models. *Journal of Machine Learning Research*, *7*.

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

Falkenauer, E. (1998). On method overfitting. *Journal of Heuristics*, *4*.

Gent, I. P. (1999). A response to "On method overfitting". *Journal of Heuristics*, *5*, 109–111.

Kalyanakrishnan, S., & Stone, P. (2009). An empirical analysis of value function-based and policy search reinforcement learning. *AAMAS '09: Proceedings of the 8th international conference on Autonomous agents and multiagent systems*. To appear.

Lane, T., & Smart, W. (2005). Why (PO)MDPs lose for spatial tasks and what to do about it. *Proceedings of the ICML 2005 Workshop on Rich Representations for Reinforcement Learning*.

Leyton-brown, K., Pearson, M., & Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. *In ACM Conference on Electronic Commerce* (pp. 66–76).

Mitra, S., Datta, S., Perkins, T., & Michailidis, G. (2008). *Introduction to machine learning and bioinformatics*. Chapman & Hall.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2004). Inverted autonomous helicopter flight via reinforcement learning. *Proceedings of the International Symposium on Experimental Robotics*.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Technical Report CUED/F-INFENG/TR 166). Cambridge University.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044).

Tesauro, G. (1994). TD-gammon, a self-teaching backgammon program achieves master-level play. *Neural Computation*, *6*, 215–219.

UMass (2009). Reinforcement learning repository at UMass, Amherst : Random MDP generators. `http://www-all.cs.umass.edu/rlr/domains.html`.

Wilson, A., Fern, A., Ray, S., & Tadepalli, P. (2007). Multi-task reinforcement learning: a hierarchical bayesian approach. *ICML '07: Proceedings of the 24th international conference on Machine learning* (pp. 1015–1022).