In Proceedings of the 16th International FLAIRS Conference, pp. 361-365, St. Augustine, FL

Learning from Reinforcement and Advice Using Composite Reward Functions

Vinay N. Papudesi and Manfred Huber

Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, 76019-0015
{papudesi, huber}@cse.uta.edu

Abstract

Reinforcement learning has become a widely used methodology for creating intelligent agents in a wide range of applications. However, its performance deteriorates in tasks with sparse feedback or lengthy inter-reinforcement times. This paper presents an extension that makes use of an advisory entity to provide additional feedback to the agent. The agent incorporates both the rewards provided by the environment and the advice to attain faster learning speed, and policies that are tuned towards the preferences of the advisor while still achieving the underlying task objective. The advice is converted to "tuning" or user rewards that, together with the task rewards, define a composite reward function that more accurately defines the advisor's perception of the task. At the same time, the formation of erroneous loops due to incorrect user rewards is avoided using formal bounds on the user reward component. This approach is illustrated using a robot navigation task.

Introduction

In recent years, learning from reinforcements (Watkins 1989 and Kaelbling, Littman, and Moore 1996) has received substantial attention as a mechanism for robots and other computer systems to learn tasks without external supervision. In this framework, the system learns from its interaction with the environment and a task-specific reward function that provides feedback about its performance at a given task. The agent typically receives reinforcement from the environment only after it achieves a particular goal or sub-goal. This intermittent and sometimes incomplete feedback makes learning from reinforcements slow. Feedback that is of a more continuous nature can make learning faster. For instance, if additional rewards are assigned to sub-goals instead of a single final reward, the task can be learned faster since sub-goals are often simpler to learn. However, it is generally non-trivial to construct reward functions that mirror the properties of every task. Often, the translation of a task into a single numerical function requires simplifications that might result in a control policy that does not correctly address the task.

Another approach to improving the performance of reinforcement learning algorithms is the incorporation of external advice. In many practical task domains there exists an external advisor, be it a human expert, a software agent or a control system, that is at least partially knowledgeable of the task to be learned. Such an advisor could aid the agent in learning the optimal control policy, as illustrated in Figure 1. The learning agent derives its control policy based not only on the reinforcement signals but also on the instructions provided by the advisor.

The advantages of learning from advice have been recognized almost 45 years ago by John McCarthy in his proposal of the *advice taker* (McCarthy 1958). (Clouse 1996) applies advice from a training agent that, if taken, yields some reinforcement as determined by the training agent regardless of the true feedback from the environment. Similarly, (Lin 1992) suggests the use of advice in the form of a sequence of actions that yield some predetermined reinforcement. Placing the onus of computing such reinforcements on the advisor could produce unpredictable (and potentially incorrect) policies and, in the case of human advisors, be a tad inconvenient. (Maclin and Shavlik 1994) describes an approach to learning from reinforcements and advice that uses knowledge-based neural networks. Advice is applied as weight-changes, thereby producing a modified policy.

This paper presents an approach to learning from reinforcements and advice in which the set of external

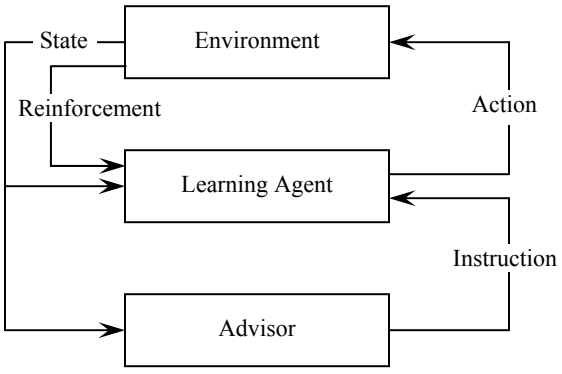

Figure 1. Learning from reinforcements and advice

instructions is used to sculpt a reward function that more accurately represents the advisor's view of the task, resulting in the learning of a potentially different control policy. Instructions serve two purposes—the acceleration of the learning process and the tuning of the reward function, which can alter the manner in which the task is solved, and possibly the task itself.

In this paper, a robot navigation task will be considered. Here, reinforcement learning and advice will be used to learn paths to target locations in a hybrid topological/ geometric path planning framework. Policies are learned in a connectivity graph where nodes represent significant locations, referred to as *via-points*, and connected nodes can be reached using geometric control primitives.

Simple and Composite Reward Functions

The construction of an appropriate reward function for a specific task is generally difficult. Often, the agent derives a large positive reward when it reaches an accepting state. In the example navigation task, a task reward function would assign a fixed positive reward to the goal states. A similar navigational task might also include a set of via-points which the robot must avoid. This information can be captured by assigning negative rewards to these states.

More complex reward structures can be created by assigning rewards to intermediate states or state-action pairs. For instance, a particular navigation task might assign a positive reward to actions that represent a passage through a door or corridor. This provides the robot with a heuristic similar to "doors and corridors are a means of reaching distant goals." Incorporating this information into the reward function can be seen as fine-tuning the policy. Negative rewards play a similar role.

Adding intermediate rewards and punishments to tune a policy is generally non-trivial and can easily result in the formation of a policy that does not perform the task correctly. One way of countering this involves determining upper and lower limits for the reward that may be applied to each state-action pair such that it does not lead to the formation of loops. However, some tasks require the formation of loops. For instance, the optimal policy for a robot that secures and patrols a building and reports when it detects a burglar would involve repeatedly cycling through the rooms of the building. It would therefore seem appropriate to differentiate between the task reward function and the "tuning" reward function, referred to within this text as the *user* reward function. These two components together define the reward function, where the task reward function is fixed *a priori* while an external advisor can define the user reward function. Throughout learning, the system ensures that cycles are not created due to modified user rewards.

The Task of Navigation

Composite rewards are used to learn the sample task of mobile robot navigation. It is important to note that while this paper develops and applies the technique of learning from reinforcements and advice in the context of navigation, it is also applicable to tasks that may be correctly described by reward functions.

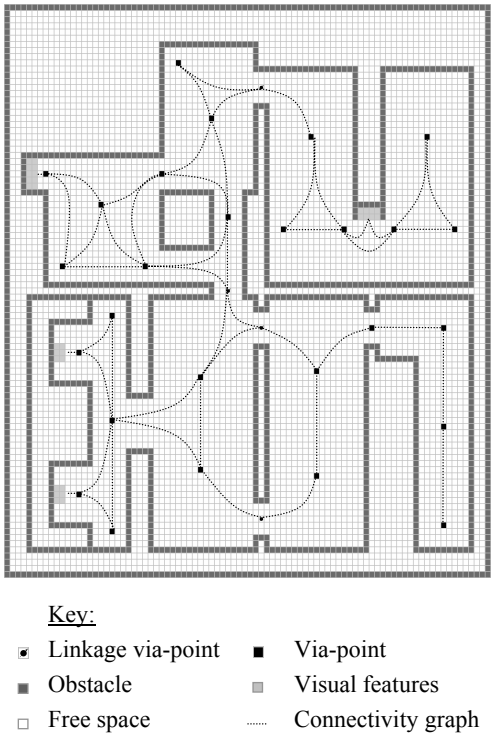

Figure 2. Representation of a sample environment

The task assigned to the mobile robot is to learn to optimally navigate to a specific target. The environment consists of a set V of via-points on a set W of maps, linked together through a subset of V representing the *linkage via-points*. Each map is a 50×50 grid of square cells[1]. Figure 2 shows a sample environment consisting of four maps or 100×100 cells. The via-points are linked together to create a connectivity graph that identifies valid paths.

Actions are specified as instances of a geometric motion controller with a particular goal. Goal specifications here consist of subsets of via-points representing absolute locations or locations of recognizable visual features. These actions directly handle the continuous geometric space by computing a path to one of the given goal locations, if such a path exists.

The abstract state space S of the learning agent is the set of via-points V. The maximum number of controllers in this state space is $2^{|S|}-1$, as the set of goals G for each controller can include any combination of the $|S|$ states. The number of controllers can be reduced to $|S|$ by restricting the goal of each controller to a single via-point. This set of controllers represents the action space A.

[1] The choice of a 50×50 grid is related to the use of harmonic potential-based motion controllers (Connolly and Grupen 1992, Connolly and Grupen 1996) to traverse via-points.

The target of a specific navigation task can be a particular via-point in V or a visual feature from the set of recognizable visual features F. The size of the set F and the complexity of the visual features depend on the capability of the robot's camera and image processing modules. Targets represented by via-points are directly reachable by at least one controller. However, controllers are only applicable from a limited number of states. Targets represented by visual features are reachable through their closest via-points. When the robot detects a visual feature at a particular via-point, it stores the via-point–feature pair in the model for future reference.

Learning from Reinforcements and Advice

The steps involved in taking advice from external entities were defined (Hayes-Roth, Klahr and Mostow 1980) as:

1. Request the advice,
2. Convert the advice to an internal representation,
3. Convert the advice into a usable form,
4. Integrate the reformulated advice into the agent's current knowledge base, and
5. Judge the value of the advice.

The first step decides what entity initiates the provision of advice. This paper assumes that the advisor initiates the provision of advice. This assumption is based on convenience for human advisors.

Often, instructions are provided in a form that is comprehensible to only the advisor (typically a human) and requires the reformulations made in steps 2 and 3. In the example presented here, the advice itself is in the form of states and state-action pairs. A recommendation of the state-action pair (s, a) leads to a preference for action a in state s. A recommendation of the state s leads to a preference for the path of state-action pairs starting at the current state and ending at s. The set of recommended states and state-action pairs is mapped onto a numerical function, $bias : S \times A \times (V \cup F) \to \aleph$. If n_s represents the total number of actions available at state s and $T \in (V \cup F)$ represents the task, *bias* is defined as:

$$bias_T(s,a) \leftarrow \begin{cases} bias_T(s,a) + n_s - 1 & \text{If } a \text{ is the recommended action,} \\ bias_T(s,a) - 1 & \text{If } a \text{ is an alternative action,} \\ bias_T(s,a) & \text{Otherwise.} \end{cases}$$

This definition of the *bias* function ensures that the numerical bias applied to each state-action pair is zero if all n_s actions at state s are recommended equally often. The notion behind this construction is that an advisor that recommends all possible actions at a state is most likely unaware of the true best action(s).

Step 4 varies with the learning approach and internal representation scheme used. In this work, the *bias* function is transformed into the user reward component $r_{\upsilon,T}(s,a)$ of the composite reward function. This transformation is defined as:

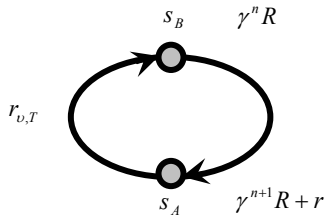

Figure 3. A simple loop.

$$r_{\upsilon,T}(s,a) = \begin{cases} f_1\left(s,a,T,\omega^*(bias_T(s,a))\right) & \text{If } bias_T(s,a) \text{ is positive,} \\ f_2\left(s,a,T,\omega_*(bias_T(s,a))\right) & \text{If } bias_T(s,a) \text{ is negative,} \\ 0 & \text{Otherwise.} \end{cases}$$

Here, ω^* and ω_* are weighting functions. Their purpose is to vary the applied user reward with the user bias. For example, consider the following instances of f_1 and ω^*:

$$f_1(s,a,T,\omega^*) = k\omega^* \quad \text{and} \quad \omega^*(x) = \frac{x}{1+x}$$

The value of the weighting function ω^* increases with the user bias towards an asymptotic value of one. If the user bias is positive, the applied user reward f_1 would be in the range $[k/2, k)$.

Finally, in step 5, the agent must analyze how the external advice has altered its performance, and accordingly retain, alter or dispose of it. A typical example is the avoidance of erroneous loops. In the work presented here, analytic limits on the user rewards are derived to avoid such misleading advice.

The User Reward Function

The user reward component associated with a particular state-action pair is directly related to the *bias* function. The sign of the bias indicates if the state-action pair has been recommended or discouraged, while the magnitude indicates how often the state-action pair has been recommended or discouraged.

Upper Bound for User Rewards

An important decision is the numerical value of the user reward. To preserve the task and speed up learning, it has to be ensured that the value is not so high as to lead to the formation of self-sustaining loops. Figure 3 illustrates such a case. The following analysis assumes a task reward function with a single final task reward and no intermediate rewards or punishments. It further assumes that the action selection mechanism is deterministic and based on the Q-value function, where $Q(s,a)$ represents the value of a state-action pair as the expected sum of discounted future rewards if action a is taken in state s and the optimal policy is followed afterwards (Watkins 1989). However, the results also apply to probabilistic models.

State s_B is closer to the task goal T than s_A and has a higher value. The Q-value of the state-action pair (s_B, a^{s_A})

after convergence is $\gamma\, Q_T(s_A, a^{s_B})$, where γ is the discount fac-tor and the notation a^t represents an action that leads to state *t*. A loop is formed if this value is larger than the value of state s_B. To ensure that no loop forms, the following inequality must hold:

$$\gamma\left(\gamma^{n+1}R + r_{\upsilon,T}\right) < \gamma^n R$$

Simplifying this inequality and solving for $r_{\upsilon,T}$ results in:

$$r_{\upsilon,T} < \left(1-\gamma^2\right)\gamma^{n-1}\, R$$

This upper bound for the user reward is correct under the assumption that no other state-action pair receives a reward. In general, both the task and user reward components may have positive and negative intermediate reward assignments. In this more general case, an upper bound for the user reward can be can be derived as [2]:

$$r_{\upsilon,T}(s,a) < (1-\gamma)\; Q_T(s,a) - r_{\tau,T}(s,a)$$

This limit is used to judge the advice in order to prevent the formation of spurious loops and thus to preserve the original task objective.

Lower Bound for User Rewards

Consider a value function that is strictly increasing. Negative rewards have the effect of reducing the values of the predecessor states. Certain assignments of negative rewards can invalidate this property. For example, any cycle of states can be made to form a loop by assigning sufficiently large negative rewards to all actions that leave the cycle. If there are no rewards within the loop, the values associated with the state-action pairs that form the loop decay to zero. If there are negative rewards within the loop, the values of the corresponding state-action pairs decay to a fixed negative value that is a function of the negative rewards and the discount rate. However, if every state has at least one action that is assigned a non-negative reward, then the values of state-action pairs in the loop decay to zero. A lower bound for user rewards can therefore be computed as the value that causes the value of the best action to become zero. When the user reward drops below this limit, the corresponding values become zero or less, and (potentially) lead to the formation of loops. In the presence of intermediate task rewards, a lower bound for the user reward is:

$$r_{\upsilon,T}(s,a) > -\max_{b\in A} Q_T(s,b) - r_{\tau,T}(s,a)$$

The Learning Algorithm

An algorithm for learning from reinforcement and advice is shown in Table 1. It learns an optimal control policy for a task *T*.

The algorithm makes dynamic programming updates on the *Q*-value and user reward functions. The *Q*-value of each state-action pair (s,a^t) depends on the probability that

[2] A detailed derivation of this bound can be found in (Papudesi 2002).

Table 1. Learning from reinforcement and advice

LEARN-WITH-USER-REWARDS (task *T*):
Initialize:
$i \leftarrow 0$
For each state-action pair (s,a^t), do $r^i_{\upsilon,T}(s,a^t) \leftarrow 0$
Repeat forever:
$i \leftarrow i+1$
For each state-action pair (s,a^t), do:
1. Determine the composite reward $r_T(s,a^t)$:
   a. Compute $r_{\upsilon,T}(s,a^t)$ according to $bias_T(s,a^t)$
   b. $r^i_{\upsilon,T}(s,a^t) \leftarrow r^{i-1}_{\upsilon,T}(s,a^t) + \alpha_\upsilon\left(r_{\upsilon,T}(s,a^t) - r^{i-1}_{\upsilon,T}(s,a^t)\right)$
   c. $r_T(s,a^t) \leftarrow r_{\tau,T}(s,a^t) + r^i_{\upsilon,T}(s,a^t)$
2. Compute the components of $Q_T(s,a^t)$:
   a. $\delta_{succ}(s,a^t) \leftarrow r_T(s,a^t) + \gamma\, Q_T\left(t, \arg\max_{b\in A} Q_T(t,b)\right)$
   b. $\delta_{fail}(s,a^t) \leftarrow \gamma\; Q_T(s,a^t)$
3. $Q_T(s,a^t) \leftarrow P(t\mid s,a^t)\;\delta_{succ}(s,a^t) + \left(1-P(t\mid s,a^t)\right)\delta_{fail}(s,a^t)$

action a^t leads to state *t*. The correct value of this probability, $P(t\mid s,a^t)$, is discovered as the agent explores the state space. An incremental user reward update scheme is essential for oscillation-free convergence of the user reward function (Papudesi 2002).

Experimental Results

This section presents the application of the LEARN-WITH-USER-REWARDS algorithm. The quadruple of user reward transformation functions used to obtain these results are:

$$f_1(s,a,T,\omega^*) = \omega^*\left((1-\gamma)\; Q_T(s,a) - r_{\tau,T}(s,a)\right),$$

$$f_2(s,a,T,\omega_*) = \max\left(\omega_*,\; -\max_{b\in A-\{a^t\}} Q_T(s,b) - r_{\tau,T}(s,a)\right),$$

$$\omega^*(x) = \frac{x}{1+x} \quad \text{and} \quad \omega_*(x) = \frac{x}{10}$$

The weighting function ω^* for positive user biases tends towards one as the user bias increases. On the other hand, ω_* causes the user reward to decrease by 0.1 with each recommendation of a different state-action pair. The user reward functions f_1 and f_2 enforce the upper and lower bounds to prevent the formation of loops.

Figure 4 shows the different paths taken by the agent. The initial and final via points are represented by the circle and cross, respectively. Without any external advice, the agent takes the shortest path to the target which is path (*a*). When the state-action pair (s,a^t) (marked as 1 in the figure) is recommended by the advisor, path (*b*) is preferred, as a result of the newly introduced user rewards.

The prevention of loops is shown by issuing a second instruction that recommends the state-action pair (t,a^s), marked as 2 in Figure 4. The agent now takes path (*c*) and avoids the formation of the expected loop. Figures 5 and 6

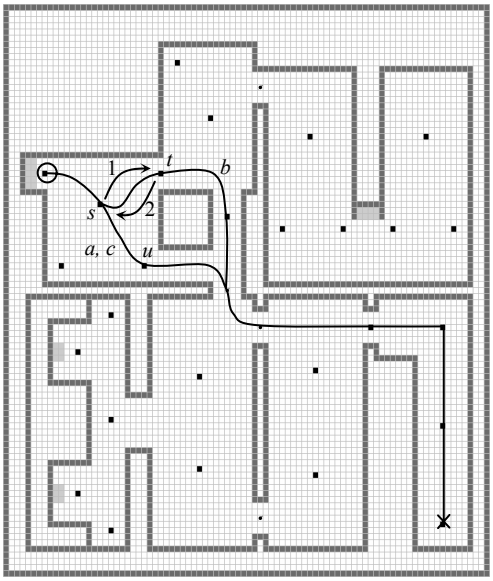

Figure 4. Paths taken by the agent with and without advice.

illustrate the changes in the user reward and *Q*-value functions due to the two instructions.

Advice intended to tune the policy may be used to specify the preferred path to a via-point. However, the upper and lower bounds on the user rewards prevents advice from changing the task itself. Experiments to this effect were successfully performed.

The learning process is generally accelerated by the provision of advice, the exception being advice that is intended to modify or tune the policy. The actual speed-up in the learning process is derived in (Papudesi 2002).

Conclusion

The results presented in this paper illustrate the potential of learning from reinforcement and advice using user rewards. By incorporating advice into an additional reward function, its independence from the task is ensured. As a result, the advisor is provided with a high degree of freedom in shaping the control policy, but cannot prevent the achievement of the overall task. Furthermore, strategically provided advice can accelerate the learning process, while incorrect advice is ultimately ignored, as its effects diminish over time.

Acknowledgments

This work was supported in part by NSF ITR-0121297.

References

Clouse J. A. 1996. Learning from an Automated Training Agent. In *Weiβ, G. Sen, S.*, Adaptation and Learning in Multiagent Systems, Springer Verlog, Berlin.

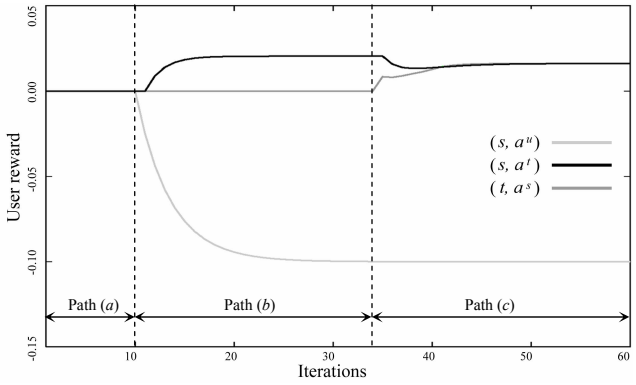

Figure 5. Changes in the user reward function due to advice.

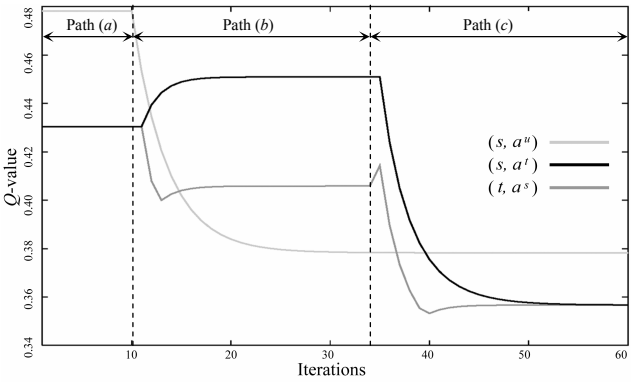

Figure 6. Changes in the *Q*-value function due to advice.

Connolly, C. I., and Grupen, R. A. 1996. Harmonic Control. In *Proceedings of the 1992 International Symposium on Intelligent Control*.

Connolly, C. I., and Grupen, R. A. 1992. Applications of Harmonic Functions to Robotics. In *Proceedings of the 1992 International Symposium on Intelligent Control*.

Hayes-Roth, F., Klahr, P., and Mostow, D. J. 1980. Advice-taking and knowledge refinement: An iterative view of skill acquisition. Technical Report, Rand Corporation.

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

Lin, Long-Ji 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293-321.

Maclin, R., and Shavlik, J. W. 1994. Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 694–699, MIT Press.

McCarthy, J. 1958. Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, volume 1, 77–84, London.

Papudesi, V. N., 2002. Integrating Advice with Rein-forcement Learning. M.S. thesis, Department of Computer Science, Univ. of Texas at Arlington.

Watkins, C. J. C. H. 1989. Learning from delayed rewards. PhD thesis, Psychology Department, Univ. of Cambridge.