Similarity in Reinforcement Learning

Peng Zang and Charles Lee Isbell Jr.

Georgia Institute of Technology {pengzang,isbell}@cc.gatech.edu

Abstract. A well-constructed similarity function is crucial to the successful application of case-based methods. We explore what a good similarity function means for reinforcement learning problems. The key observation is that the cases one is likely to reach due to one's future actions are the most relevant. We demonstrate this idea using a simple algorithm that we test in two domains against standard techniques.

1 Introduction

Reinforcement learning (RL) is a broad field that defines a particular framework for specifying how an agent can act to maximize its long-term utility in a stochastic (and perhaps only partially observable) environment. RL approaches typically focus on learning a *policy*: a reactive plan that accomplishes specific goals (expressed implicitly through a reward signal). A common approach is to learn a *value function* that captures the true utility of being in a given state. Such approaches have been applied successfully to a variety of problems in prediction and control, including riding a bicycle and playing Backgammon.

Unfortunately, many techniques fail to scale to large problems. Faced with large state spaces, it becomes necessary to approximate the value function. Instance based methods, *e.g.,* case based methods, are often used [1–3] because they can, for example, easily learn a particular subspace and perform well in that subspace without ever observing, learning, or representing the rest. They also tend to be more stable in terms of convergence [4].

As with all instance-based techniques, a good similarity function is crucial. One typical assumption used in constructing such a function is that syntactic similarity in the input space implies similarity in the output space.

To understand whether this is a reasonable assumption, it is useful to understand what properties make a similarity function "good". Abstractly, a good similarity function is one that must, at a minimum, identify cases that are *relevant* to generating an optimum solution for an input. In particular, *relevant* cases are those whose solutions are highly dependent; that is, if we treat the solution to a case as a random variable, A, then another case is *relevant* if the solution to the first is not independent of the solution to the second, B: $P(A,B) \neq P(A)P(B)$. Contra-positively, if the solutions to two cases are independent—knowing one solution does not affect the probability of the other—then the two cases are irrelevant to one another.

In this paper, we focus on learning approximate value functions in sequential decision problems addressed by RL. We will do so by identifying relevant states without use of the assumption that syntactic similarity in input space implies similarity in the output space, as such an assumption is often inaccurate. The key observation is that the states one is likely to visit in the future from a particular state are relevant. We demonstrate this idea using a simple reinforcement learning algorithm that we test in two domains.

2 Problem Definition

Following [4], we model the sequential decision problem as a finite Markov decision process (MDP). Formally, an MDP $M = (S, A, P^a_{ss'}, R^a_{ss'})$ is defined by a set of states S, a set of actions A, a transition model of the probability of reaching state s' by taking action a in state s $P^a_{ss'} = Pr(s_{t+1} = s' | s_t = s, a_t = a)$, and a reward matrix $R^a_{ss'}$ specifying the immediate reward received when taking action a in state s and reaching the new state s'.

A policy, $\pi(s)$, is a mapping dictating what action an agent should perform in a particular state. The utility or *value* of a state $V^\pi(s)$ is the expected long-term (typically discounted) rewards an agent receives assuming it follows policy π from state s. V^* is the value assuming an optimal policy that maximizes the long-term expected reward. Similarly, $Q^\pi(s, a)$ refers to the expected long-term reward for taking action a in state s and then following policy π. $Q^*(s, a)$ is defined appropriately for the optimal policy.

Techniques to solve MDPs typically revolve around the value function V mapping states to their expected long-term rewards. Unfortunately, the number of states grows exponentially with the number of dimensions. Methods that use a direct, tabular representation of V do not scale. A common technique to address this problem is do use some functional form to approximate the value function. In this paper we are focused on using case based approaches to approximate the value function V. A case will consist of simply the state (the problem) and the value it is approximating (the solution).

3 Similarity in RL

Consider navigating a simple, discrete 2D maze (see Figure 3). States are simply the squares in the maze, each state represented by a row and column pair. The agent is represented by the dark blue square and the goals by light green squares. We will assume some low uniformly random negative reward for state transitions to encourage reaching the goal as quickly as possible. In this scenario, states inside the sealed room on the right are useless in estimating the value of the state where the agent is located. This is intuitive to grasp. Values of states in the sealed room on the right are dependent solely upon the rewards defined for states in that room. Thus the values for states inside the room will all be 10, the reward for reaching the goal in the room, minus some small amount for getting to the goal. In contrast, outside the room, the only reachable terminal state yields a reward of 100 and so values of all outside states will be dependent upon that. Not only are the values of states inside and outside the room different,

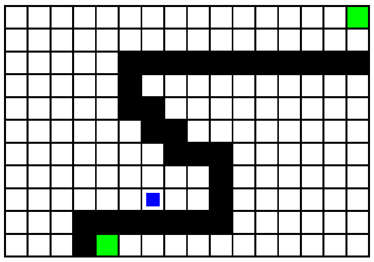


Fig. 1: Motivating maze example

but in fact, they are completely independent. Knowing the value of a state on the other side of the wall does not and cannot help in estimating the value of a state on this side.

This simple example illustrates the general point that similarity in the input space often does not translate to anything useful about the output space. Common similarity measures based on L_1 or L_2 distance would retrieve irrelevant cases and perform poorly. What then might be a good similarity function?

Recall that good cases are those whose solutions are highly dependent with the solution of the problem at hand. The insight is that while there are many sources of statistical dependence, the policy defines a successor relation between states and thus defines a particularly important dependence. The bellman equation [4] that holds over the values of states describes this succinctly: $V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V^{\pi}(s'))$. This equation guarantees that states one is likely to reach in the future due to the dynamics of the world and the actions one will take have solutions that are highly related to the solution of the current state. Further, we can extract from the bellman equation the heuristic that the further in the future a state s' is, the more complex the relationship between it and s. This stems from the simple observation that the bellman equation describes one-step relationships and must be "unrolled" many times for states many steps in the future. Assuming reward and transition models of some complexity, the more levels unrolled, the more complex the relationship and the more difficult it will be to leverage. All of this points to a method of finding good cases: simply identify states one is likely to come across in the future starting with the closest.

4 Case-based Approximate Policy Iteration (CAPI)

We present CAPI, a simple approximate policy iteration algorithm using a cased-based function approximator. CAPI serves as the vehicle with which to test our method for finding relevant cases. In particular, CAPI locates cases through a limited size forward search projection.

4.1 The CAPI Algorithm

CAPI alternates between generating trial runs (policy improvement) according to the current policy, and training on those trial runs (policy evaluation). We use an ϵ-greedy policy to select the action maximizing $\sum_{s'}[P^a_{ss'}(R^a_{ss'} + \gamma\hat{V}(s'))]$ with probability $(1-\epsilon)$, where γ is a discount factor. A random action is selected with probability ϵ^2, and an intentfully exploratory action leading to a less-visited state is selected with probability $\epsilon(1-\epsilon)$. Training adds examples into the case base, thereby updating the policy induced by $\hat{V}$. These trial runs can also be used to learn a model $M = (P^a_{ss'}, R^a_{ss'})$ if one is not provided. This iteration continues until the desired convergence threshold is met.

4.2 Case Based Value Function Approximation

In CAPI, a case is stored for each encountered state along with the estimated value of that state. Given the transition and reward models, case base C, and a query state q, the approximated value $\hat{V}(q)$ is derived by:

1. **Retrieve.** In order to find states we are likely to encounter in the future we use the model to perform a forward search of fixed-size. Instead of simply biasing the searching based on likelihood of reaching a state however, we use the likelihood multiplied by the expected reward of reaching that state to favor likely reached states with high reward (or low cost). This reflects the fact that we wish to estimate V^* as opposed to V^π as the policy π constantly changes as we gather more experience in the world.
2. **Reuse.** Instead of simply interpolating between the values of the cases retrieved, we can do something more intelligent. Because we performed a forward search to locate relevant cases, we have the paths between q and the retrieved cases as well as the expected reward along those paths. This allows us to use the path rewards to compute exact estimates. Further, we know we are performing maximizing backups with the estimated value as part of policy improvement, *i.e.,* estimating V^*. Thus instead of using the average of the estimates different cases report, we use the maximum.
3. **Revision and Retention** Revision and retention is handled by the policy evaluation step. The value inferred by our case based learner is used to select the appropriate action for our agent to take. At the end of an episode—assuming the episode ends in a terminal state (rather than by exceeding some maximum number of trial steps or timeout)—a full backup updating $\hat{V}$ is performed along the trajectory from the terminal state back to the start state. This update gives us the revised value estimate for each case (state) encountered during the episode. Whenever a backup encounters a state already in the case base, the retention strategy is to replace the old with the new if and only if the new value is higher. Because we perform full backups from terminal states, we know that the value we back up for each state is at least a lower-bound on the true value. Similarly, the full backups from subsequent runs also produce lower-bounds on the true value.

5 Analysis

CAPI is an off-policy algorithm meaning there is no restriction as to how cases are generated. In particular, it means that although our implementation uses an epsilon greedy version of the current policy to generate cases, there is no general requirement to do so and cases could conceivably come from a random policy. CAPI bootstraps, as it updates estimates for states using other learned estimates without waiting for the end of an episode. If the transition model is known to be accurate, we can initialize the value of all states to some minimal reward and prove convergence. In particular, if the maximum episode length is k and the minimum reward r_{min}, $k \cdot r_{min}$ is a sufficient minimal reward and allows us to ensure that at any point in time, $V_k(s)$ reflects a lower bound on the optimal value function $V^*(s)$.

Proof. $\forall k V_k(s) \leq V^*(s)$

Base case:

$$\begin{aligned} V_0(s) &= \min(\{R^a_{ss'} | a \in A \text{ and } s, s' \in S\}) \\ &\leq V^*(s) \end{aligned}$$

Inductive case:

$$V_{k+1}(s) = \max(V_k(s), \max_a \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V_k(s'))) \tag{1}$$

By inductive hypothesis we know $V_k(s) \leq V^*(s)$. Further, the Bellman optimality equation [4] tells us:

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V^*(s'))$$

Thus $\max_a \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V_k(s')) \leq V^*(s)$.

Because both arguments in (1) are less than or equal to $V^*(s)$, $V_{k+1}(s) \leq V^*(s)$.

By accepting only value updates that increase our estimated value, $V_k(s)$ must be monotonically increasing until $V_k(s) = V^*(s)$. Because CAPI is ϵ-greedy, all state-action pairs will be visited an infinite number of times as the number of episodes approaches infinity. Thus the lower bound will monotonically converge to the true utility.

If the model may be inaccurate (*e.g.,* we must learned it as we go), updates must be made incrementally. Instead of only accepting updates that are higher, all updates have an incremental effect: $\hat{V}(s) \leftarrow (1-\alpha)\hat{V}(s) + \alpha\hat{V}'(s)$ where $\hat{V}(s)$ is the previous estimate, $\hat{V}'(s)$ is the new estimate, and α is a learning rate. As long as α is reduced reasonably[1], convergence should be maintained.

[1] For example, $\sum_{t=1}^{\infty} \alpha_t(a) = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2(a) < \infty$ where a refers to the action

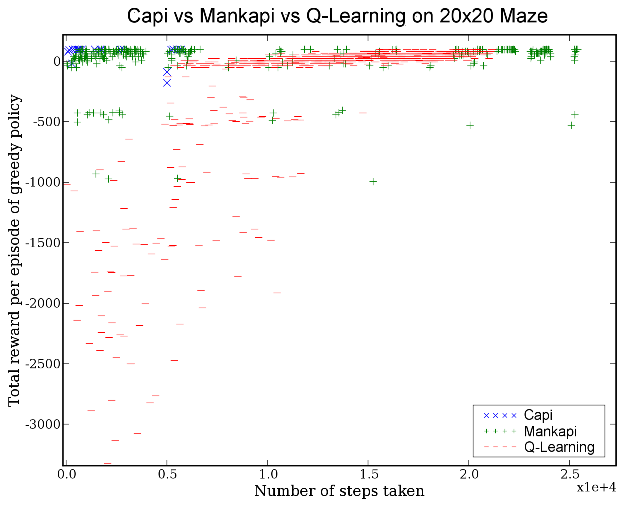
6 Experiments

We are interested in using CAPI to empirically verify the validity of our case retrieval method. As such we compare CAPI against two other algorithms. The first, ManCAPI, is identical to CAPI except that it is based on a k-nearest neighbors approach and as such uses Manhattan distance to retrieve cases and interpolation in its reuse step to estimate the queried value. ManCAPI allows us to explore how our method compares to traditional distance based retrieval methods. We also compare to standard Q-learning as a baseline for RL performance. All convergence results are averaged over 10 runs. Average total reward per episode is determined by making the policy greedy, holding it fixed, and averaging the total reward received over 10 random episodes. We ran experiments in two different domains and performed comparisons both in terms of data efficiency as well as training time.

6.1 Maze domain

The first domain is a standard grid world with obstacles in the form of non-lethal mines. Mines are distributed uniformly random, the number of mines set as a percentage of the number of states in the grid world (twenty percent). Four actions, *up*, *down*, *left*, and *right* are available everywhere except when the agent would move off the grid. The starting location is a uniformly random non-mine, non-goal state. The state is represented simply by the agent's coordinates. All state-actions have a reward of -0.1 except those that take you into a mine (-10) and those that take you to the goal location ($+100$).

To evaluate data efficiency, we compare how quickly the various algorithms converge to an optimal policy as a function of the number of steps taken in the world. Figure 6.1 shows the convergence results for CAPI, ManCAPI and Q-Learning in a $20x20$ maze. CAPI is the fastest to converge on average, followed by Q-Learning. Most interesting is the performance of ManCAPI. Even though Manhattan distance is a reasonable similarity function for grid world domains, ManCAPI still performs the worst. Though not shown, several ManCAPI runs took over 70,000 steps before converging. In cases when ManCAPI converges quickly, it is typically when the mines are placed such that they do not form a large barrier to the goal and degrade the Manhattan distance estimate. When that is not the case, the similarity function pulls poor cases and the resulting estimate actually worsens the performance of the agent. This can be likened to falling into a local minima, and demonstrates the importance of a good similarity function. We also compared CAPI and Q-Learning in a 50x50 maze (ManCAPI was too slow to converge on a maze of this size). We expected CAPI to be more data efficient (require fewer steps in the world) than Q-Learning because it uses a function approximator, whereas Q-Learning has no information in unexplored areas. Using the function approximator however, does mean a projection is required to determine the greedy action so we expected CAPI might be slower. Surprisingly, CAPI actually took on average about the same number of steps



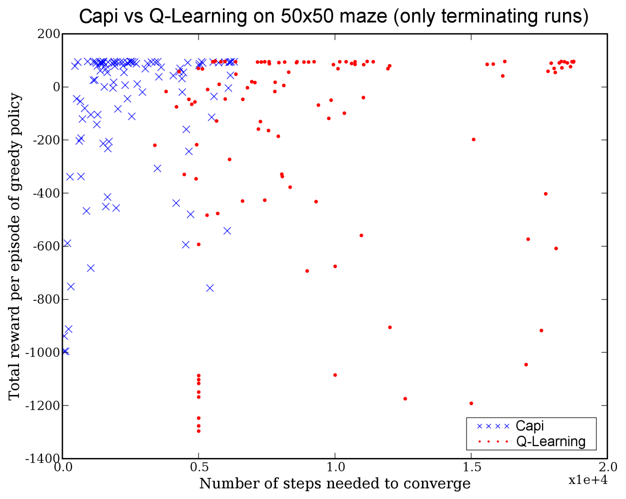
(11,000) to converge as Q-Learning. Even more surprisingly, the slower convergence in terms of data efficiency did not amount to slower convergence in time. CAPI is actually faster in terms of time needed to converge, taking on average 35 seconds compared to Q-Learning's 45 seconds.

This seemingly contradictory result can be explained when we examine CAPI's behavior more closely. The reason CAPI takes more steps is because until an episode ends in a terminal state (rather than by exceeding a maximum number of trial steps), no training examples are generated. This means nothing is learned in non-terminating episodes, so the case base remains empty. In a large state space with a low density of terminal states, this can persist for quite a while. In some of our trials, the case base remained empty and learning stalled, until well after 20,000 steps have been taken in the world.

If we remove episodes in which terminal states are not reached and no training data is generated, we obtain a better picture of how fast CAPI learns (see Figure 6.1). We can see in these graphs that CAPI converges much more quickly than Q-Learning. On average, CAPI converges after 3,000 steps versus the 11,000 steps Q-Learning requires. This demonstrates the effectiveness of our case based function approximator and by extension, our case retrieval mechanism. We expect an online formulation of CAPI that generates cases not just at the end of terminating episodes but during episodes whether terminating or not, to be able to avoid the stalled learning problem.

6.2 "Baby Toy" domain

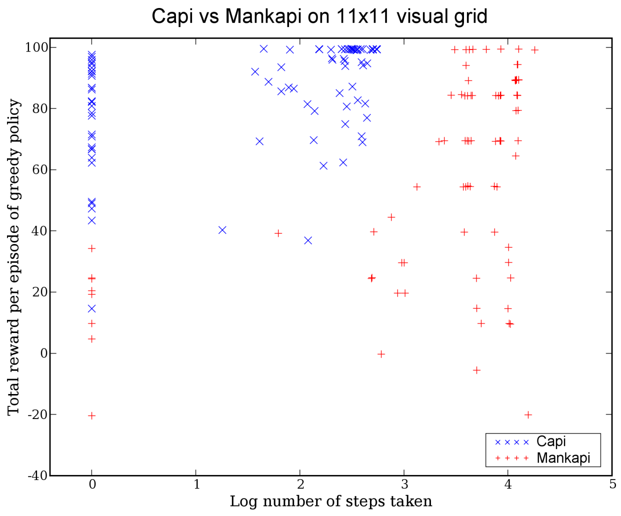
Our second domain is the "Baby Toy" world, modeled after a toddler's game. The agent is given blocks of different shapes and a box with correspondingly-shaped openings. The goal is to push the blocks into the box, but this can only be done through the appropriate hole. This world is modeled visually: the state



is a black and white rasterized bitmap. At each state, the agent selects how to move the block: *up*, *down*, *left*, *right*, *clockwise*, *counter-clockwise*, or *push*. The *up*, *down*, *left*, and *right* actions move the block by one pixel, while *clockwise* and *counter-clockwise* rotate the block in 30-degree increments. The shape of a block, its location and orientation are randomly chosen at the start of each episode but constrained so that 30 degree rotations allow a solution. All actions have a reward of -0.1 except the *push* action yields a reward of $+100$ when successful and -10 otherwise. Figure 6.2 shows a graph comparing the data efficiency of CAPI and ManCAPI. CAPI converges roughly two orders of magnitude faster. The difference is so large in this domain because Manhattan distance makes for a poor similarity function. It will often confuse shifted triangles for squares, rotated triangles for shifted squares, and so on. For example, a 30-degree rotation on a triangle can flip more pixel values than replacing the triangle with a square block. Not only does this mean that the case base must be extremely dense before Manhattan distance retrieves useful cases, but it also means that in the early stages of learning, poor actions are selected which often lead the agent astray such that it cannot reach a terminal state. CAPI in contrast, suffers no such problems. It is able to retrieve relevant cases and approximate the value function effectively. This also makes CAPI fast, typically reaching the optimal policy before the fastest of Q-Learning runs begins to converge. Finally, we also performed scaling experiments over differently sized maps. CAPI scales about as well as Q-Learning but is about an order of magnitude faster.

7 Related Work

The key focus of our work is finding relevant cases for RL problems in a data-dependent way. There is similar work in learning similarity metrics or shaping



neighborhoods [5], [6], [7]. These approaches optimize similarity metrics based on data, often using cross-validation procedures (leave-one-out). This optimization is over the free parameters (*i.e.,* weights) of a pre-specified similarity function, typically vector-based distance in input space. One still must choose a class of similarity functions and if an inappropriate choice is made, performance will suffer. In contrast, our similarity function is not based on syntactic analysis of the input space, but works based on the dynamics of the world and the policy.

If one examines CAPI mechanistically, it is very similar to RTDP [8]. Our particular instantiation of CAPI could be considered a variant on trial-based RTDP. This is coincidental. The CAPI approach is conceptually quite different: given a slightly different revision or retention strategy (*e.g.,* not keeping a case for every state) the resulting version of CAPI would be quite different from RTDP. Some of the extra optimizations in CAPI are common. For example, our use of updating backups from projected neighbors is similar to Samuel [9]. His checkers-playing program used backups from projected, hypothetical states in the future. Similarly, using experience and the model to guide exploration has been discussed in [10]. In any event, we emphasize that the specific implementation is tangential. It serves primarily as a vehicle for empirically validating our idea for retrieving relevant cases based on likely future outcomes, which is the main thrust of this paper.

8 Conclusion

In this paper we explored what makes similarity functions "good". In particular, we defined *relevance* as statistical dependence and applied it to sequential decision problems. In that setting, we found at least one way of feasibly retrieving similar cases: finding those one is likely to encounter in the future because of one's policy and the dynamics of the world.

We implemented CAPI as a proof of concept to empirically explore our proposed retrieval method. As the experiments show, CAPI performs significantly better than a more traditional L_1 similarity based method as well as Q-Learning, demonstrating the effectiveness of our retrieval method.

While CAPI validates that useful cases are the ones we are likely to encounter in the future, the implementation of the retrieval does not scale. This is because CAPI must perform projections to identify and retrieve those cases. A fixed radius projection is exponential in the number of dimensions. Thus what we really desire is a method of "compiling down" the projections into a similarity function that is fast. This is the focus of our future work. One technique we are exploring involves performing a number of projections initially, but then using those as training for a supervised procedure for developing fast predictors of projections.

References

1. Tadepalli, P., Ok, D.: Scaling up average reward reinforcement learning by approximating the domain models and the value function. In: International Conference on Machine Learning (ICML). (1996) 471–479
2. Aha, D.W., Salzberg, S.L.: Learning to catch: Applying nearest neighbor algorithms to dynamic control tasks. In: Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics. (1993) 363–368
3. Gabel, T., Riedmiller, M.: Cbr for state value function approximation in reinforcement learning. (2005)
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
5. Stahl, A.: Learning feature weights from case order feedback. In: Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR), London, UK, Springer-Verlag (2001) 502–516
6. Lowe, D.G.: Similarity metric learning for a variable-kernel classifier. Technical Report TR-93-43 (1993)
7. Peng, J., Heisterkamp, D.R., Dai, H.: Lda/svm driven nearest neighbor classification. (2001) 58–64
8. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artif. Intell. **72**(1-2) (1995) 81–138
9. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal on Research and Development (1959) 210–229
10. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning **13** (1993) 103–130
11. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. (2003) 1107–1149
12. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. Artificial Intelligence Review **11**(1-5) (1997) 75–113
13. Cleveland, W.S.: Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association **74** (1979) 829–836