An Empirical Analysis of RL's Drift From Its Behaviorism Roots

Matthew Adams
North Carolina State University
mbadams3@ncsu.edu

Robert Loftin
North Carolina State University
rtloftin@ncsu.edu

Matthew E. Taylor
Lafayette College
taylorm@lafayette.edu

Michael Littman
Rutgers University
mlittman@cs.rutgers.edu

David Roberts
North Carolina State University
robertsd@ncsu.edu

ABSTRACT

We present an empirical survey of reinforcement learning techniques and relate these techniques to concepts from behaviorism, a field of psychology concerned with the learning process. Specifically, we examine two standard RL algorithms, model-free SARSA, and model-based R-MAX, when used with various shaping techniques. We consider multiple techniques for incorporating shaping into these algorithms, including the use of options and potential-based shaping. Findings indicate any improvement in sample complexity that results from shaping is limited at best. We suggest that this is either due to reinforcement learning not modeling behaviorism well, or behaviorism not modeling animal learning well. We further suggest that a paradigm shift in reinforcement learning techniques is required before the kind of learning performance that techniques from behaviorism indicate are possible can be realized.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Theory, Performance

Keywords

Markov decision processes, reinforcement learning, behaviorism

1. INTRODUCTION

Human-canine collaboration is unique and astonishingly successful. It is an existence proof that human beings can communicate complex tasks to non-human autonomous agents. With only minimal communication tools, humans and dogs can work together to play fetch, hunt, search for buried avalanche survivors, track missing persons, sniff for explosives/narcotics/contraband, or guide the blind. Published literature on training going back to Skinner's work on behaviorism [8] provides insight into how dogs learn, and how human trainers teach dogs complex tasks quickly and accurately.

At a very high level, machine reinforcement learning agents act in a way similar to animals undergoing training. They take actions from different states, and their behavior over time is affected by the rewards that they are given. In this paper, we report on a series of experiments designed to illustrate how the performance of two common machine reinforcement learning paradigms compares to the experiences human trainers have when training dogs. Specifically, we examine two popular reinforcement learning techniques: the model-free SARSA algorithm [9] and the model-based R-MAX algorithm [11]. We performed a survey of popular dog training methods methods and identified computational analogues to those approaches. We ran experiments to evaluate whether reinforcement learning would benefit from "shaping" inputs commonly used in behaviorism-inspired dog training techniques. We were looking for significant performance increases where learning would occur in orders of magnitude less time or for problems to be learned that weren't learnable without these techniques.

Although a few of our experiments resulted in faster learning rates for the agent, the benefits that were seen were nowhere near the improvements in learning rates seen in animals when corresponding training techniques are used. At best, applying shaping techniques to reinforcement learning agents resulted in about half as many atomic actions taken in order to learn an optimal policy. This pales in comparison to the effects shaping has on animals.

2. CANINE LEARNING

Shaping by successive approximation [13] is a technique used in animal training by which desired behaviors are taught to learners by means of selectively rewarding actions closer and closer to the desired behavior. At first, in order to obtain a reward, the animal only needs to perform an action that is vaguely similar to the desired one. Once the animal learns how to get the reward, the criteria is tightened and the animal has to perform actions that are more similar to the desired behavior in order to obtain a reward. Eventually, the criteria converges and the goal behavior is learned.

Clicker training is a popular paradigm for shaping by successive approximation in animal training [7]. First, the positive stimulus of a treat is paired with the click sound from a noise-making device. This process of charging the clicker makes it a "conditioned secondary reinforcer" using classical conditioning [6]. Because the click is associated with a treat, the click noise indicates a promise of a future reward. In order to train the dog to perform a certain behavior, a click is given whenever the dog's actions get closer to the target behavior. After every click, a primary reinforcer (*e.g.*, a treat) is provided and the episode is over. The criteria for clicking is tightened as the dog more closely approximates the target behavior.

Shaping by successive approximation in animal training allows animals to learn much more rapidly than if the same rewards were given only when the exact target behavior was performed. Without

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

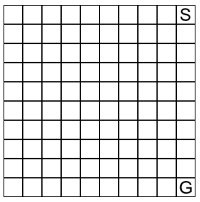

Figure 1: The 10x10 open grid with goal locations.

any cue as to the sort of behavior the animal is supposed to be doing, the animal can only behave arbitrarily until it finally stumbles into performing the behavior. The addition of the shaping rewards expedite the process by guiding the animal into the sort of behavior it is supposed to perform.

There are many ways that a trainer can use shaping concepts. The specifics of when to use what approach are part of the "art" of training. That being said, there is overwhelming evidence that these approaches enable efficient learning of complex tasks that wouldn't be realistic to learn without shaping (*cf.*, [3, 7, 8]). We believe that in order to get machine reinforcement learning to perform in some of the awe-inspiring ways we see human-canine teams perform, the algorithms must respond to the behavior shaping process in an appropriate way. The experiments we describe in subsequent sections of this paper illustrate that, unfortunately, RL algorithms do not respond favorably to shaping.

3. EXPERIMENTAL DESIGN

Reinforcement learning problems are related to Markov decision processes (MDP). A MDP is defined by a tuple $\{S, A, T, R\}$, with S the set of possible states, and A the set of actions which the agent can take. The transition function $T : S \times A \times S \mapsto [1:0]$ represents the probability of transitioning to state s' after performing action a in state s, and the reward function $R : S \times A \mapsto \mathbb{R}$, determines the reward received for performing action a in state s.

The goal of the learner is to find a policy $\pi : S \mapsto A$, that maximizes the expected discounted total reward that the learner receives. Model-based algorithms such as R-MAX learn explicit models of T and R, whereas model-free methods such as SARSA learn a function $Q : S \times A \mapsto \mathbb{R}$, which represents the expected discounted total reward for taking action a in state s. If the state and action spaces are discrete and of sufficiently small size, the transition and reward models, or the Q function, as well as the policy π, can all be represented as tables. For large or continuous state spaces, some form of function approximation, such as a neural network or tile-coding, is often used to represent these functions. For the purposes of this paper, we focus on tabular representations only. Our goal is not to scale these algorithms to large problems, but to see how their performance on smaller problems changes when different behavior shaping techniques are used in the learning process.

In order to determine the ability of current machine learning algorithms to operate in paradigms similar to and display performance characteristics similar to human-dog teams, we conducted a series of simulation experiments on an open grid world. At each timestep, the agent can move one step in any of the four cardinal directions, and the result of each action is deterministic. The goal state is in one corner of the grid, and reaching the goal state results in a reward of 1 and the end of the episode. All other states have a reward of zero. The discount factor was 0.8.

The initial state for each episode is in an adjacent corner to the goal state (see Figure 1). Defining the problem this way, rather than having the initial state and goal state on opposite corners, creates a region of the state space that will not be reached by applying an optimal policy, and therefore is not likely to be fruitful to explore.

One of the major advantages of using the simple grid space is the incremental adjustments that we can make to the size of the state space. The same dynamics of the space exist when the region is small (*e.g.*, 10x10) or large (*e.g.*, 100x100), so good comparisons can be made between methods on varying problem sizes.

Other reasons to use the grid space include the fact that the optimal policy is known and is easy to visualize. Because we know what the optimal policy should be, we can compare it to the policy that the agents actually learn and investigate how long it takes the agents to approximate the optimal policy.

The simplicity of the grid space allows us to see how the agent reacts to changes in the problem in accordance with what humans might do when training animals. Many reinforcement learning methodologies and improvements that have been investigated in the past are only applicable to learning domains with very particular properties. The basic grid space can accommodate many modifications that allow us to simulate the different sorts of training modifications that are done in animal training.

By using this simple environment, we were better able to compare the various algorithms' performance under different conditions. We sought to determine: ***Does the performance of learning algorithms improve in ways consistent with canine learners when canine training techniques are used during machine training?*** To that end, we tested both a model-based learning algorithm (R-MAX) and a model free learning algorithm (SARSA) in various conditions modeled after common canine training techniques. For each of these algorithms, we were looking for a *significant* improvement in performance when common variants of canine training techniques were applied to the machine training process. By "significant," we mean criteria commonly used in evaluating canine behavior: speed of learning and accuracy of learned behaviors. Due to the simplicity of our experimental domain, we focused entirely on learning speed in the survey.

These two learning algorithms were chosen because they are characteristic of the two major approaches to reinforcement learning problems. SARSA is a good representative of model free methods, whereas R-MAX is a good representative of model-based methods. Applying shaping techniques to both methods gives us a good survey of how reinforcement learning methods react in general to canine training techniques.

The SARSA Algorithm: The SARSA algorithm is an on-policy temporal difference reinforcement learning method. The algorithm builds a value table using the SARSA update rule, which updates values using the action that will be taken at the next step, as opposed to the greedy action. We used a learning rate $\alpha = 0.1$ and optimistic Q-value initialization combined with a pure exploit policy. In part, we did this because it is a good model of the behavior we observe in canines. They receive an "intrinsic" reward from the environment, but a more valuable reward from the trainer. Once a dog understands what behaviors cause rewards, it will relentlessly exploit the reward until something better comes along [4].

The R-MAX Algorithm: The R-MAX algorithm [11] estimates a model of both the MDP transition probabilities and reward function. These models are initialized to assume that all actions in all states deterministically transition to the fictitious state G_0, and receive a reward of R_{max}. The algorithm keeps a record of what transitions and rewards have been observed for each state and action. Initially, and whenever a new state-action pair becomes known, the algorithm computes an optimal policy under the current model. In our implementation this is done using value iteration with a convergence threshold of 0.1. The problem that the R-MAX agent learns

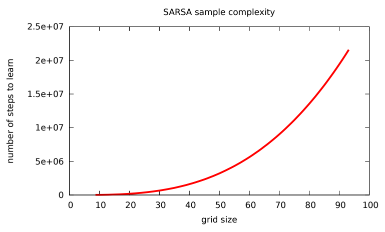

Figure 2: The number of steps taken to learn using SARSA as a function of the grid size.

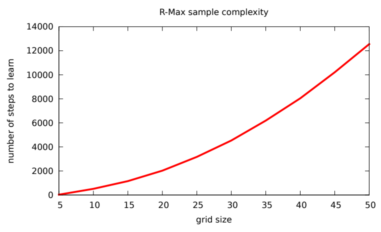

Figure 3: The growth rate of the number of actions required for R-MAX to learn the optimal policy, as a function of the width and height of the square grid.

on is exactly the same as for the SARSA agent, with rewards of 1 for the goal state and 0 for all other states. As the agent acts greedily, the parameter R_{max} is set optimistically to 1.0 to encourage exploration, and its value affects the degree to which the learner exploits its current knowledge. As the considered domains are fully deterministic, a state-action pair is considered known when it has been observed once.

Algorithm Performance: To be able to make solid empirical comparisons of the effects of various shaping techniques, it is critical to have good metrics to compare the learning rates of the agents. However, in order to compute metrics about how long it takes the agent to learn a problem, it is necessary to make a decision about when the agent has sufficiently learned a good policy for the problem. One common solution is to wait until the stored value of Q-estimates converges, and no changes are made during an episode that are greater than a certain threshold. Because this method can result in the agent learning far after a decent policy has been established, we decided to use a method that was more strictly performance based. Because we know the optimal policy for the grid space, we can determine how many steps the agent needs to take to get to the goal in the ideal case. In order to declare that the agent has learned the problem sufficiently, it needs to get to the goal in four out of the five most recent episodes within 105% of the steps the optimal policy would result in. This so called 80% criteria is a common criteria used by dog trainers [2].

Unfortunately, passing this competence criteria does not guarantee that the agent has a stored policy that leads exactly to the goal. If the agent updates its policy during the last episode before it is declared finished, the update can change the policy to something invalid. In order to make sure this does not happen, we need to explicitly verify the policy before the trial is declared finished. To do this, we simply walk through the policy from the start state without making any policy updates. If the goal state is not reached, then we let the agent continue learning until it passes both the competence metric and the policy is deemed valid.

We opted to focus on measuring performance of the agent using the cumulative number of steps required across all episodes in order for the algorithm to learn a policy according to the 80% criterion described above. There are other metrics we could have examined. For example, we could have measured the number of episodes needed to learn the policy, the quality of the resulting policy, or the CPU time. However, as all of these measures are related in some way and vary highly depending on the particular learning algorithm, there was little insight we could gain beyond what was available using the steps measure.

4. BASELINE PERFORMANCE

To establish a baseline for performance, we ran both algorithms on the open grid. For SARSA, we scaled the grid size from 10x10 to 100x100. As shown in Figure 2, the number of steps taken to learn a policy in accordance with the competence criteria grew rapidly with respect to the size of the state space.

In order for a shaping method to show adequate improvement over the baseline SARSA methodology, the results need to show either a reduction of the growth rate of steps in accordance with the state space size, or cut the learning time by a significant constant factor. Some shaping methods might trade higher or lower numbers of episodes for shorter or longer episodes, so comparing the number of atomic actions taken to learn an adequate policy is a good comparative metric for the shaping methodologies.

For R-MAX, we scaled the grid from 5x5 to 50x50, with the start and goal states in the same positions as in the SARSA example. The results are presented in Figure 3. The number of action steps required to learn the policy grows roughly linearly in the number of states (quadratically in the grid dimension). Because in this case the algorithm only needs a single example to learn the model for a given state-action pair, and since R-MAX can compute an optimal policy as soon as it has a complete model, it makes sense that the sample complexity would scale in this way. As a model-based method, R-MAX performs much better in terms of sample complexity than model-free SARSA. However, it should be noted that R-MAX incurs a significant computational cost, as it must repeatedly plan a new policy. The complexity of computing this policy also scales with the size of the state space, but is dependent on the details of how that policy is computed. While this makes direct comparison with SARSA difficult, here we are concerned with how sample complexity can be improved for an algorithm relative to itself using different training paradigms.

5. POTENTIAL-BASED METHODS

One common approach to shaping behaviors in canines is to use a reward as a lure to instruct the desired behavior. For example, in this “lure-reward” paradigm [3], if you were trying to teach a dog to “heel” (stand at your side, facing forward, shoulders aligned with your knees) you might do so with a treat as a lure. You would begin by placing the treat in front of the dog’s nose, and moving it a few inches toward your side. When the dog moves forward to grab it, you give it to her. This is repeated, moving the lure closer to your side until she is in position. This process of providing intermediate rewards for each step towards the ultimate goal, is akin to reward shaping in machine learning [5].

5.1 Potential Bands

Reward schemes based on potentials are a traditional method in reinforcement learning to introduce the idea of shaping behavior. Each state in the state space is associated with a potential value, and the reward given for any action is simply the difference in potentials of the resultant state and the original state. By changing the potential values of states, the characteristics of the agent’s learned

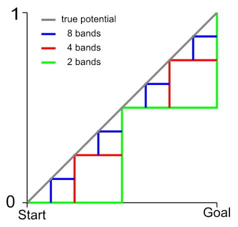

Figure 4: A depiction of various potential bands. As the number of bands grows, the resulting potential function is an increasingly accurate approximation of the true potential.

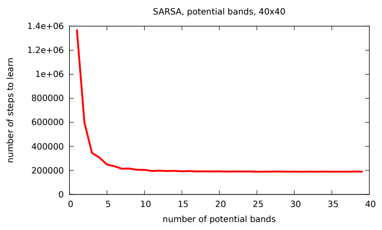

Figure 5: The growth rate of the number of actions required for SARSA to learn the optimal policy on a 40x40 grid, as a function of the number of distinct *potential bands*.

policy can be changed.

A simple way to use potentials for shaping by successive approximation is to group states according to how far away they are from the desired goal state, and then to assign potentials based on how far the group is from the goal. In this way, the groups of states are different approximations of the desired behavior, for which closer and closer approximations are preferred.

We set the potential for the goal state to 1, and the potential for the state farthest from the goal state to zero. We then equally space bands of potentials between those two states. The potential value for each state in that band is:

$$\Phi(state) = 1 - \frac{dist(band, goal)}{dist(start, goal)}, \tag{1}$$

where $dist(x, y)$ is the distance between either a state or band x and another state y. The potential values for the bands thus increase regularly as the states get closer to the goal.

Figure 4 contains an illustration of how "bands" are used to estimate the true potential function. As seen in Figure 5, as one might expect, the more bands of potentials that were used, the faster that the agent was able to learn how to get to the goal state—more potential bands equate to a more accurate approximation of the true potential of each action. Using more bands entailed narrow bands and decreased potential differences between the bands. As the number of bands increases, the average distance that the agent has to move before it gets a reward drops. With all of this extra information, the agent learns much faster. With as few as 10 bands on a 40x40 grid, with a width of each band of 4, the agent using SARSA learns at nearly its maximum pace.

In the case with the maximum number of potential bands, the agent receives a positive reward whenever it moves closer to the goal, and a negative reward any time it moves away. As one would expect, it learns extremely rapidly in this scenario. In the degenerate case with just one band, the problem is identical to the basic case, and the agent learns at the basic rate.

However, these potential bands do not correspond very well with

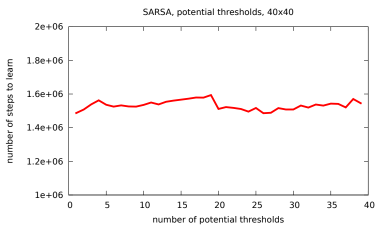

Figure 6: The growth rate of the number of actions required for SARSA to learn the optimal policy on a 40x40 grid, as a function of the number of *potential thresholds* used.

how dogs are generally trained. Dogs generally don't get feedback at every point that it gets closer or farther from the desired behavior. Even in the lure-reward paradigm, the rewards are spaced out over time. As our goal is to investigate how the agents react when canine training techniques are used, it makes sense to try a different method for using potentials to better simulate those techniques.

5.2 Potential Thresholds

In training methods that try to shape by successive approximation, the animal is rewarded when its behavior approximates the desired behavior within a certain "distance". When the animal demonstrates proficiency on behavior within a given approximation, the threshold is made tighter so that the animal has to get even closer to the desired behavior. For example, when teaching a dog to "watch me," they are first asked to look for just a few seconds. Once they are proficient, the duration of maintaining eye contact is expected to increase (a new threshold).

A more accurate way to shape agent behavior by successive approximation using potential shaping is to use a threshold function for the potential value. For simplicity and consistency, the values of our function are zero and one.

$$\Phi(state) = \begin{cases} 0 & : dist(state, goal) > threshold \\ 1 & : dist(state, goal) \leq threshold \end{cases} \tag{2}$$

Once the agent crosses the boundary between the potentials and receives a reward, the episode terminates. Once the current threshold has been learned sufficiently, the criteria is changed and a lower threshold (closer approximation to the desired behavior) is used.

In order to train the agent to reach the goal state, we need to decide when it is appropriate to up the criteria and move the threshold closer to the goal. We use the same criteria inspired by dog training to decide whether or not the agent has sufficiently learned the problem as a whole to decide whether or not the agent has learned the subproblem [2]. In order to move on to the next potential threshold, the agent has to cross the threshold boundary within 105% of the minimum steps four out of the five most recent trials (the 80% rule). Again, because criteria changes are determined during learning (and not in a separate evaluation process) a policy verification step is necessary to make sure a Q-table update during the last trial did not make the policy invalid.

As shown in Figure 6, the performance for the agent using potential thresholds did not change much. The number of steps taken to learn the policy in accordance with the competence criteria was fairly consistent with the base SARSA algorithm (illustrated in Figure 7) regardless of the number of thresholds used. Thus, despite starting with easier tasks and slowly increasing complexity (*i.e.*, by using more thresholds), RL algorithms perform essentially the same. This is because, regardless of the number of potential thresholds, the world must be explored to build the Q-table.

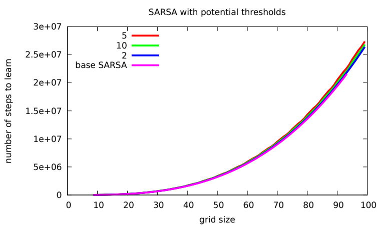

Figure 7: The growth rate of the number of actions required for SARSA to learn the optimal policy, with 2, 5, and 10 *potential thresholds*, as a function of the square grid size.

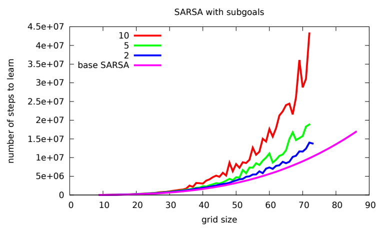

Figure 8: The growth rate in the number of actions required for SARSA to learn the optimal policy for varying numbers of subgoals, as a function of grid size.

6. SUCCESSIVE APPROXIMATIONS

A paradigm that is gaining significant momentum in the dog training world is clicker training [7]. Through the use of a conditioned secondary reinforcer [6] humans are able to more effectively shape the behavior of canines by requiring them to perform increasingly more accurate approximations of the desired behavior. Shaping, conceived of this way, is a process by which a common start state is used (*e.g.*, the dog is in a standing position) and the goal state is moved closer and closer to the ultimate goal (*e.g.*, the dog's behind is closer and closer to the floor for a sit behavior). In this section, we report on a variety of experiments that use this shaping by successive approximations technique.

A straightforward method of performing successive approximation is to introduce the idea of subgoal states. Rather than giving a reward for the agent reaching the ultimate goal state, we give a reward once the agent has reached some state that lies along the optimal trajectory between the start state and the goal state.

For a given optimal trajectory between a start state and a goal state, we choose a number of subgoals spaced evenly between the start state and the goal state.[1] We begin by running episodes starting from the same start state, but where a reward is given and the episode is terminated when the agent reaches the first subgoal.

We determine when the agent has sufficiently learned how to get to the subgoal by the same competence criteria as used in the previous experiments. When the agent has learned how to get to a particular subgoal, the subgoal that is next closest to the goal state becomes the current subgoal and is the only state that is rewarded.

Once the agent has demonstrated that it knows how to get to the final subgoal, the true goal state becomes the state that is rewarded. As in the original problem, the agent must get to the goal state within 105% of the minimum distance to the goal four out of five of the most recent trials and the policy must be verified to lead to the goal in order for the agent to be done. Our primary metric of concern was the total number of steps taken. The total number of episodes used to learn is an interesting measure, but use of the subgoal architecture is likely to result in a shorter average episode length because the start and goal states are closer.

It is worth noting that any time the criteria changes and a new subgoal is selected, the old Q-values become inaccurate because the reward is non-stationary. The Q estimate for *all* actions of states close to the subgoal will be fairly close to the reward value, as they are only a few states away from getting a reward. When the Markov decision process is changed so that the subgoal is farther away, all of those values will need to decrease, rendering the work done to give preference to the action along the optimal trajectory futile.

Even worse, the policy for states that are between the subgoal and the goal state are likely to point back to the previous subgoal even when the subgoal state is moved closer to the goal. Time is wasted when the episode does not terminate when the agent reaches the previous subgoal, and the policy for states closer to the goal pushes the agent backwards. That portion of the policy has to be unlearned before the agent can move on to reaching the subgoals closer to the goal. The data we present support these observations.

6.1 Simple Subgoal Decompositions

In our testing we found that SARSA with optimistic initialization and a fixed exploit policy was frequently unable to learn an optimal policy in the face of non-stationary rewards. Because certain Q-values that should reflect a reward for later subgoals in the decomposition may be learned to have low Q-values during training on earlier subgoals in the decomposition, the learner may have a hard time converging on more accurate higher values when the subgoal criteria changes. In those cases, learning may get stuck. Therefore, to better compare the results of using subgoals to the basic agent, we used an epsilon-greedy action selection policy for the agent that used the simple subgoal decomposition. The agent takes a random, exploratory action 5% of the time, which fixes the issue by allowing the agent to discover the fact that the optimal trajectory has a higher value than the one that is stored.

Figure 8 shows that the performance of agents using the subgoal architecture was consistently worse than agents learning on the base problem. The more subgoals that are included in the architecture of the problem, the longer that the agent takes to complete the problem. The growth rate of the number of steps that the agent takes versus the number of subgoals used is not directly proportional, which points to the fact that there is some benefit gained to having the knowledge of how to get to the previous subgoal over starting from scratch on each subgoal state. However, this benefit is dwarfed by the amount of overhead taken up by the subgoal architecture, either by proving that the agent knows a policy to get to the subgoal well enough, or by relearning parts of the policy that differ once the subgoal is moved.

As seen in Figure 9, similar problems can be seen when using R-MAX with subgoals. To incorporate subgoals into R-MAX, we attempted to consider all subgoals simultaneously. The transition and reward models for the current subgoal are updated to reflect the fact that the episode terminates there, and are set such that any action causes the agent to remain in the subgoal with 0 reward. When any subgoal is reached and it is determined that the current policy is optimal for that subgoal, then the subgoal is removed. When a subgoal is removed, the model is reset to show all actions from that state leading to state G_0 with reward R_{max}. When a transition or reward does not match the current model, the value for

[1]Although omitted from this paper, we found that the spacing of subgoals had no effect on performance. This is actually contrary to what we observe in dog training, where subtask complexity can increase as learning persists. We believe that curriculum design [12] is actually crucial to algorithm performance in the long run.

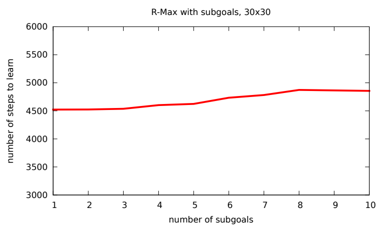

Figure 9: The growth rate of the sample complexity for R-MAX using subgoals, as a function of the number of subgoals

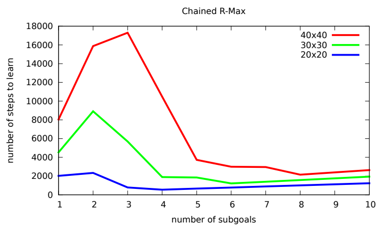

Figure 10: The growth rate of the sample complexity of R-MAX, as a function of the number of subtasks.

that transition is replaced, and the optimal policy is recomputed. This allows the agent to adjust its model to reflect the removal of subgoals. Once the reward for an earlier subgoal was reduced, the new greedy policy would be exploratory until the new (sub)goal was learned. However, because there were still many unexplored states when the subgoal was moved, the exploration policy would not always be efficient at finding the new goal. As a result, it took longer to find each new subgoal, as well as the final goal, than it would have taken without the subgoals.

For the R-MAX algorithm, subgoals lead to unnecessary exploration, and so we attempt to address this issue by decomposing the full task into subtasks which should require less exploration to solve. In this case, we consider moving from one subgoal to the next as distinct subtasks that are learned separately by R-MAX, and the final policy then consists of a chain of policies from the start state, through each subgoal, to the final goal. Each subtask is processed in order going towards the goal, but as is the case for the Q-tables for options, each subtask is learned independently, without sharing transition or reward models. Learning on a subtask completes when an optimal policy for that task has been found.

Figure 10 shows that with a sufficient number of subtasks, that is, with subtasks that are sufficiently small, a lower sample complexity can be achieved. For a smaller number of subtasks, however, the number of steps required actually increases. There is a specific number of subtasks for each grid size at which the sample complexity peaks and then begins to decrease. With a sufficient number of subtasks the total sample complexity of solving all of the subtasks is actually less than that of solving the full task. One likely explanation for this is that once the number of subgoals becomes large enough, and consequently the distance from one subgoal to the next drops below a certain threshold, the optimal policy at the earlier subgoal is to move towards the next subgoal, instead of trying to reach unexplored states, which it assumes yield reward R_{max}, but now incur a greater penalty to reach than the subgoal. Chained R-MAX does show a substantial improvement in performance, in terms of sample complexity, over the base R-MAX. There is, however, a point after which sample complexity begins to increase again as a function of the number of subgoals,

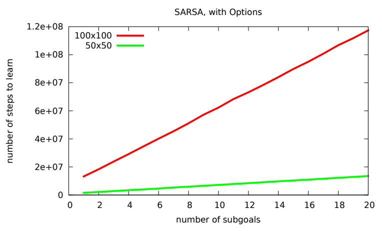

Figure 11: The growth rate of the sample complexity using options, as a function of the number of options.

suggesting that there is still a limit to how much of an improvement can be gained through this method. Further, these gains can only be realized when each subgoal is considered an independent subtask—something that does not model real-world learning.

6.2 Subgoals with Options

In order to avoid the issue of forcing the agent performing SARSA to do extra work by unlearning the Q-estimates near the previous subgoal, we can leverage the options framework [10] to try to improve shaping by successive approximation. We still start with giving a reward to the agent when it reaches the first subgoal. However, when the agent has demonstrated that it knows how to get to the subgoal according to the 80% competence criteria, instead of using the same table of Q-estimates for the following subgoal, we start with a fresh table of Q-estimates and package the previous Q-table as an option available at the start state (including recursively-defined options). The initiation set for the option is just the original start state. The option exits with probability 1.0 upon reaching the subgoal for which the option was built, and probability 0.0 otherwise. The Q-estimate for the option is initialized in the same optimistic manner as for the other actions. The agent is not forced to take the option, but is very likely to learn to do so as the option will follow a near-optimal policy towards the next subgoal.

Because the policy for each subgoal is required to pass policy verification before being declared complete, the option is guaranteed to terminate. It is possible, and very likely, that the policy that is wrapped into an option contains a previous option as part of the policy. As the problem progresses, there could be a recursive stack of options potentially as large as the total number of subgoals. Each of the options on the stack ultimately controls the behavior from its corresponding previous subgoal to its own subgoal state.

As seen in Figure 11, using options in this way increases the sample complexity over the basic algorithm, that is, the case with only 1 option. The problem caused by inconsistency between the optimal policies to reach different subgoals is avoided by using the options architecture. However, there is an additional sample overhead caused by the fact that the agent must learn, with every new subgoal, that it needs to take the provided option over the atomic actions. It seems that this overhead is still much greater than the benefits to be had by shaping the behavior of the agent with subgoals that are easier to reach.

7. LEARNING FROM EASY MISSIONS

Shaping by successive approximation, as described above, is a common tool for teaching dogs simple behaviors (like sit, stay, or down). However, it is also very common when teaching dogs more complex behaviors to use a slightly different approach. Strictly speaking, this other approach is still shaping by successive approximation. However, rather than keeping a fixed start state and varying the goal, a fixed goal is used and the start state is moved increas-

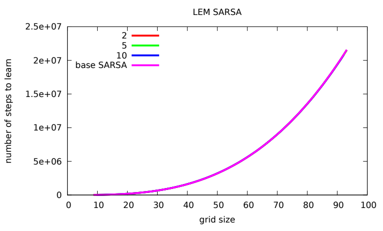

Figure 12: The growth rate of the LEM agent as the grid size scales as compared to the basic agent.

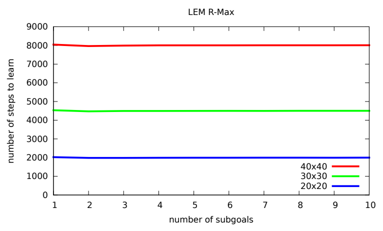

Figure 13: The sample complexity of the LEM R-MAX agent as a function of the number of start states.

ingly "farther" away. This technique is very commonly used when training working dogs (*e.g.*, hunting dogs [2]).

One of the major problems that slows down shaping by successive approximation using simple subgoals is the fact that the Q-estimates that are learned for each subgoal are inconsistent with the $Q^{\star}$ values for the optimal policy. This results from moving the goal state to perform shaping. Instead of using a complex method such as applying the options framework in order to get around this issue arising from moving subgoals, we can instead keep the goal location fixed and vary the start state (like in hunting dog training). This process is known as "learning from easy missions" in RL [1].

A set of start states are evenly spaced between the ultimate start state and the goal state. The agent starts by performing episodes starting from the start state closest to the goal. Once the agent reaches the 80% competence criteria as before and the policy is verified to lead to the goal, the start state is moved back to the next position. This process is repeated until the agent demonstrates that it knows how to get from the final start state to the goal state.

The major advantage that fixing the goal state has over the basic subgoal method is that the portions of the Q-table that are learned initially according to the early start states do not need to change when the start state is moved away from the goal state. Because the same state is always rewarded, intermediary states' values are unchanged and there is nothing unlearned when subgoals change.

The agent performed strictly better than the agent using the basic subgoals architecture (see Figure 8). However, as seen in Figure 12, like the agent that used potential thresholds, any performance gains over the baseline SARSA were limited. This result held regardless of the size of the state space or the number of start states.

Compared to the agent not using subgoals at all, there is a fair amount of sample overhead that comes with using multiple start states. A speed gain is accomplished when the agent spends more time closer to the goal and thus less time wandering aimlessly far from the goal. The most productive learning can occur just outside the region for which the agent has a good policy to get to the goal, and thus reasonable estimates for the value of the actions. Because the agent is using dynamic programming, the agent is able to expand that region outward to new states when it is on the frontier just outside of the region it knows well. With the start state moving progressively backward, the agent is able to spend a larger percentage of time on the productive frontier learning rapidly. The EVFA algorithm [14] for approximating solutions to MDP's can be said to do something similar to this. It maintains a set of states for which it has an accurate value estimate. The algorithm uses those estimates to expand the set by learning values for nearby states.

R-MAX is well suited to the LEM approach, as neither the transition function nor the reward function change when the start state is moved. In this case, the start state is moved back when an optimal policy from the current start has been found. The same transition and reward models are used for each start state, and the known/unknown parameters are not changed. As seen in Figure 13 LEM does not degrade the performance of R-MAX, it does not improve it either. As the start state is moved back, the algorithm may need to explore states that it did not need to when the start state was closer to the goal, and so the total number of actions taken is the same as if the full task were solved at once.

8. EPISODIC INTERRUPTION

One of the challenging things when training dogs is to deal with the fact that they frequently will lose interest in training. Perhaps they get full and a food reward becomes less enticing. Sometimes, especially when they are puppies, they lose focus focus and wander off. Whatever the cause, it is often necessary as a dog trainer to interrupt an errant dog and start the trial over again. Generally, this is accompanied by a verbal correction so the dog knows its actions were inconsistent with the task. By using this technique, dog trainers are able to focus their training sessions on reinforcing successful trials, thereby reducing the amount of time dogs spend performing irrelevant actions.

One observation about the performance of agents in our state space is that they spend a large portion of their time wandering in the large region away from both the start state and the goal state. The agent using SARSA and the LEM approach on a 50x50 grid space spent, on average, only 7.5% of its steps in states that are within two steps of the optimal trajectory between the start and goal state. The policy near the goal state is well-defined, and once the agent nears the goal state, the episode ends within a short period of time. However, the policy far from the goal state is nearly random, and the agent can spend a lot of time in the distant region. Additionally, little utility is gained from spending time in that region. Temporal difference learning relies on the fact that the stored value for the next state the agent will enter is a reasonable estimate for the future value of the previous state. However, the stored values in this unexplored region are mostly arbitrary, so not much useful learning can actually happen.

An additional observation is that the most rapid learning takes place on the frontier between the region near the goal state for which it has a near-optimal policy, and the region far from the goal state for which it has an arbitrary policy. If the agent is closer to the goal state, it does not improve much on the already near-optimal policy. If the agent is too far from the goal state, it wanders aimlessly without a good way to improve its policy. However, when it is on that frontier, it can expand what it knows about the region near the goal state into the area farther from the goal state.

In order to improve the performance of the algorithm, we can force it to spend more time on the productive frontier than in other regions. One way to do this is to end the episode early if the agent takes more than a certain number of steps. The number of steps is set to some amount greater than the minimum number of steps between the current start state and the goal. If the agent enters

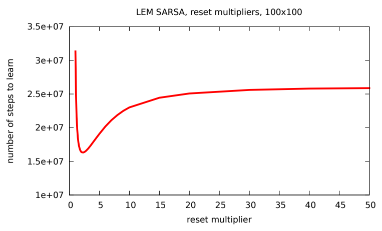

Figure 14: How the parameter determining how long to wait before resetting the trial affects how long the agent takes to learn on a 100x100 grid.

the area near the goal for which the policy is already near-optimal, then the episode will end anyway. However, if the agent enters a region far from the goal state and starts to wander, than it will get cut off before it wastes too much time wandering aimlessly. In the Learning from Easy Missions framework, the current start state is always near, but not in, the area that the agent already has a good policy for. Thus the potential advantage of giving more attempts to learn between the current start state and solidified region is large.

In order to explore the effect of resetting the episode when the agent takes too long, we ran a series of trials on grids of different sizes with different parameters for when the episode was cut off. The episode was cut off when the number of steps taken reached some multiple of the minimum steps necessary to reach the goal from the current start state along the optimal trajectory. This multiple was varied from 1.0 to 10.0.

As in shown in figure Figure 14, the performance of the agent is highly contingent upon when the episode is cut off. If the parameter is set too high and the episode is allowed to go on too long, the performance converges upward to the performance of the agent without cutting episodes off, as one might expect. However, if the parameter is set too low and the agent is cut off to early, performance rapidly deteriorates because the agent has trouble reaching the goal state before the episode terminates.

The optimal parameter for resetting the episode seems to be to reset the episode when the agent takes a number of steps around twice the minimum distance necessary to reach the goal. In this case, the agent takes about two-thirds as many steps as it would without cutting off any episodes early. In addition, on the 50x50 grid space, it spends 17.1% of its steps on average in states within 2 steps of the optimal trajectory versus 7.5% for the agent that does not use episodic interruption. Although this improvement is notable, it is not indicative of a strong change in the agent's ability to solve the problem when subgoals are introduced.

9. CONCLUSION

In this paper, we have surveyed a number of techniques for shaping behavior. We have drawn connections between these topics in the canine training literature and the machine learning literature. We have conducted an empirical analysis of these techniques on a simple, open grid environment where complete policy and performance comparisons can be made. Overwhelmingly, we found that the "behavior" of both model-based and model-free reinforcement learning algorithms did not match our expectations based on our experience training dogs and on relevant dog training literature.

Based on these data, we conclude that shaping techniques are unlikely to yield significant improvements in the performance of standard reinforcement learning algorithms in simple grid (and similar) domains. As these shaping concepts enable dog trainers to train dogs to incredibly high performance, it would be expected that any model of animal learning should see similar gains when such techniques are applied. The fact that similar improvements are not found suggests that standard reinforcement learning algorithms are insufficient as models of animal learning. It may be the case that, for certain domains and state representations, shaping techniques may prove more effective, or it may be that these learning algorithms are fundamentally unsuitable for this form of training. Regardless of which may be true, reinforcement learning—being more a dynamic programming technique than model of learning—may need a paradigm shift if it is to benefit from the training regimens that human trainers use with animal learners.

Future analyses will extend this work to stochastic domains, as well as to domains with continuous or factored state spaces, using function approximation. Additional work will examine different environments and learning tasks to determine what properties of tasks, if any, afford shaping. Future work will also contribute the development of algorithms which are more suitable to shaping.

10. REFERENCES

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 223:279–303, 1996.

[2] J. Barry, M. Emmen, and S. Smith. *Positive Gun Dogs: Clicker Training for Sporting Breeds*. Sunshine Books, 2007.

[3] I. Dunbar. *Before and After Getting Your Puppy: The Positive Approach to Raising a Happy, Healthy, and Well-Behaved Dog*. New World Library, 2004.

[4] S. R. Lindsay. *Handbook of Applied Dog Behavior and Training, Procedures and Protocols*, volume 3. Wiley-Blackwell, 2008.

[5] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[6] I. P. Pavlov. *Conditional Reflexes*. Oxford Univ. Press, 1927.

[7] K. Pryor. *Don't Shoot the Dog!: The New Art of Teaching and Training*. Bantam, 1999.

[8] B. F. Skinner. *Science and Human Behavior*. Macmillan, 1953.

[9] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[10] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[11] P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100(1-2):177–224, April 1998.

[12] M. E. Taylor. Assisting transfer-enabled machine learning algorithms: Leveraging human knowledge for curriculum design. In *Proceedings of the AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers*, 2009.

[13] E. L. Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review Monograph*, Supplement 2:1–109, 1901.

[14] P. Zang, A. J. Irani, P. Zhou, C. L. Isbell, and A. L. Thomaz. Using training regimens to teach expanding function approximators. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS10)*, pages 341–348, 2010.