Bootstrapping Memory-Based Learning with Genetic Algorithms

John W. Sheppard and **Steven L. Salzberg**
Department of Computer Science
The Johns Hopkins University
Baltimore, Maryland 21218
Email: lastname@cs.jhu.edu

Abstract

A number of special-purpose learning techniques have been developed in recent years to address the problem of learning with delayed reinforcement. This category includes numerous important control problems that arise in robotics, planning, and other areas. However, very few researchers have attempted to apply memory-based techniques to these tasks. We explore the performance of a common memory-based technique, nearest neighbor learning, on a non-trivial delayed reinforcement task. The task requires the machine to take the role of an airplane that must learn to evade pursuing missiles. The goal of learning is to find a relatively small number of exemplars that can be used to perform the task well. Because a prior study showed that nearest neighbor had great difficulty performing this task, we decided to use genetic algorithms as a bootstrapping method to provide the examples. We then edited the examples further to reduce the size of memory. Our new experiments demonstrate that the bootstrapping method resulted in a dramatic improvement in the performance of the memory-based approach, in terms of both overall accuracy and the size of memory.

Introduction

Recently, the machine learning community has paid increasing attention to problems of delayed reinforcement learning. These problems generally involve an agent that has to make a sequence of decisions, or actions, in an environment that provides feedback about those decisions. The feedback about those actions might be considerably delayed, and this delay makes learning much more difficult. A number of reinforcement learning algorithms have been developed specifically for this family of problems. However, very few researchers have attempted to use memory-based approaches such as nearest-neighbor for these problems, in part because it is not obvious how to apply them to such problems. While memory-based learning is not generally considered to be a reinforcement learning technique, it is an elegantly simple algorithm and exhibits some marked similarities to the reinforcement learning method known as Q-learning (Sutton 1988). However, as we show below, nearest-neighbor has inherent difficulties with reinforcement learning problems. One purpose of this study is to show how to overcome those difficulties and put nearest-neighbor on an equal footing with other methods.

For our study, we considered a reinforcement learning problem that was posed, in simpler form, by Grefenstette *et al.* (Grefenstette, Ramsey, & Schultz 1990). The original work showed that this task, known as *evasive maneuvers*, can be solved by a genetic algorithm (GA). In the basic problem, a guided missile is fired at an airplane, which must develop a strategy for evading the missile. In our modified problem, two guided missiles are fired at the airplane. In a preliminary study comparing nearest-neighbor (NN), GAs, and Q-learning, we found that NN was by far the worst method in its performance on this problem (Sheppard & Salzberg 1993). As a result, we sought to develop an approach that would improve the overall performance of nearest neighbor on this task.

We found that one idea was key to our success: the use of an already-trained GA to generate examples. For this task, an example is a state-action pair. Because reinforcement only comes after a long sequence of actions, it is difficult to determine which actions were good and which were not. Thus it is equally difficult to know which actions to store in a memory-based system. What we needed was some method that would increase the probability that a stored example was a good one; i.e., that the action associated with a stored state was correct. After our preliminary study showed that GAs could perform quite well on the two-missile problem, we decided to use an already-trained GA to provide the exemplars. Second, we applied a nearest-neighbor editing algorithm to the exemplar set provided by the GA to further reduce the size of the set. Our experiments demonstrate remarkable improvement in the performance of nearest neighbor learning, both in overall accuracy and in memory requirements, as a result of using these techniques.

The idea of using memory-based methods for delayed reinforcement tasks has only very recently been considered by a small number of researchers. Atkeson

(Atkeson 1989) employed a memory-based technique to train a robot arm to follow a prespecified trajectory. More recently, Moore and Atkeson (Moore & Atkeson 1993) developed an algorithm called "prioritized sweeping" in which "interesting" examples in a Q table are the focus of updating. In another study, Aha and Salzberg (Aha & Salzberg 1993) used nearest-neighbor techniques to train a simulated robot to catch a ball. In their study, they provided an agent that knew the correct behavior for the robot, and therefore provided corrected actions when the robot made a mistake. This approach is typical in nearest-neighbor applications that rely on determining "good" actions before storing examples.

Genetic algorithms have also been applied to perform delayed reinforcement problems. In addition to studying the evasive maneuvers task, Grefenstette (Grefenstette 1991) applied genetic algorithms to aerial dogfighting and target tracking. Ram applies genetic algorithms to learning navigation strategies for a robot in an obstacle field (Ram *et al.* 1994). He also applies case based reasoning in combination with reinforcement learning on the same domain (Ram & Santamaria 1993), both approaches yielding excellent performance.

Some investigators are also exploring the use of teachers to improve reinforcement learning applications. For example, Barto's ACE/ASE (Barto, Sutton, & Anderson 1983) incorporates a teaching mechanism with one connectionist network providing reinforcement to another. Clouse and Utgoff (Clouse & Utgoff 1992), who also used ACE/ASE, monitor the overall progress of the learning agent, "reset" the eligibility traces of the two learning elements when the performance fails to improve, and then provide explicit actions from an external teacher to alter the direction of learning.

The Evasive Maneuvers Task

Grefenstette *et al.* (Grefenstette, Ramsey, & Schultz 1990) introduced the evasive maneuvers task to demonstrate the ability of genetic algorithms to solve complex sequential decision making tasks. In their 2-D simulation, a single aircraft attempts to evade a single missile. The missile travels faster than the aircraft and possesses sensors that enable it to track the aircraft. The missile continually adjusts its course to collide with the aircraft at an anticipated location. The aircraft possesses six sensors to provide information about the missile, but the simulation has no information about any strategies for evasion. We initially implemented this same task, and then we extended the problem to make it substantially more difficult by adding a second missile.

In our task, the missiles are launched simultaneously from randomly chosen locations. The missiles may come from different locations, but their initial speed is the same and is much greater than that of the aircraft. As the missiles maneuver, they lose speed. Traveling straight ahead enables them to regain speed, but if they drop below a minimum threshold, they are assumed to be destroyed. The aircraft successfully evades the missiles by evading for 20 time steps or until both missiles drop below a minimum speed threshold. To make the problem even more difficult, we also assume that if the paths of the missiles and the aircraft ever pass within some "lethal range," then the aircraft is destroyed; i.e., the missiles need not collide with the aircraft. We use the term "engagement" to include a complete simulation run, beginning with the launch of the missiles and ending either after destruction of the aircraft or successful evasion of the missiles.

When flying against one missile, the capabilities of the aircraft are identical to the aircraft used by Grefenstette. In the two missile task, the aircraft has 13 sensors. When flying against one missile, the aircraft is able to control only the turn angle. When flying against two missiles, the aircraft controls speed, turn angle, and countermeasures.

Using k-NN for Evasive Maneuvering

The nearest neighbor algorithm is a classical approach to machine learning and pattern recognition, but it is not commonly used for reactive control problems. K–NN is a procedure that is typically applied to classification tasks in which a series of labeled examples are used to train the algorithm. The labels usually correspond to classes. When a new example is processed, the database of stored examples is searched to find the k examples that are closest according to some distance metric (usually Euclidean distance). The new example is assigned a class according to the majority vote of its k neighbors.

We formulated the sequential decision problems as classification problems by letting the states correspond to examples, and the actions correspond to classes. In order to be successful, a memory-based approach must have a database of correctly labeled examples. The difficulty here, though, is how to determine the *correct* action to store with each state. One can argue that we need to know the result of an engagement before deciding whether to store an example. Even after a successful evasion, though, we cannot be sure that the action at *every* time step was the correct one.

To illustrate the problems that k-NN has with the evasive maneuvering task, we briefly describe some findings of our earlier study (Sheppard & Salzberg 1993). At first, the nearest-neighbor learner generated actions randomly until the aircraft evaded the missiles for a complete engagement. The corresponding state-action pairs for that engagement were then stored. Once some examples were stored, k-NN used its memory to guide its actions. If the aircraft failed to evade when using the stored examples, it repeated the engagement and generated actions randomly until it succeeded. Not surprisingly, the algorithm sometimes took a very long time to succeed using this random

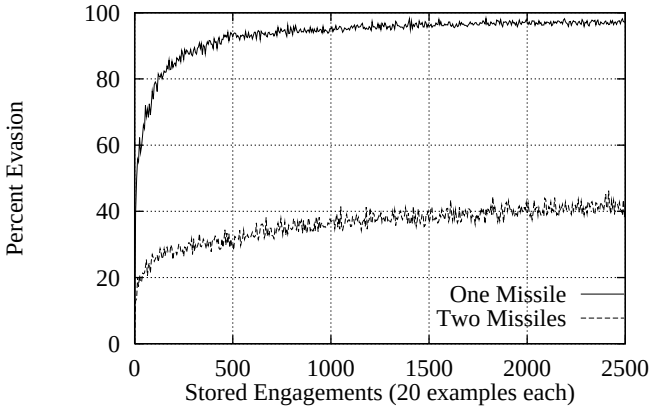

Figure 1: Performance of k-NN on evasive maneuvers.

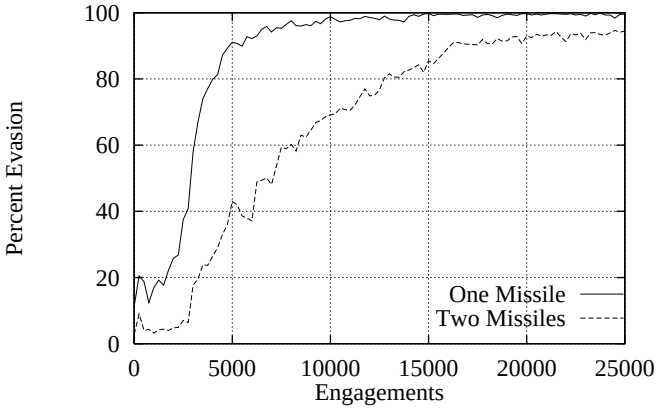

Figure 2: Performance of the genetic algorithm on evasive maneuvers.

strategy. Whenever the aircraft successfully evaded, the algorithm stored 20 examples, one for each time step.

For the initial experiments using k nearest neighbors, we varied k between 1 and 5 and determined that $k = 1$ yielded the best performance. Figure 1 shows the results of these experiments. These graphs indicate performance averaged over 10 trials for an aircraft evading one missile and two missiles. The accuracy at each point in the graph was estimated by testing the learning system on 100 randomly generated engagements.

These experiments indicate that the problem of evading a single missile is relatively easy to solve. NN was able to develop a set of examples that was 95% successful with only 10,000 examples after approximately 1,500 engagements, and it eventually reached almost perfect performance. When the aircraft attempted to learn how to evade two missiles, the results were not as encouraging. In fact, we quickly found that NN had difficulty achieving a level of performance above 45%. This indicated the two missile problem is significantly more difficult for our approach to learn.

The Genetic Algorithm

For details of our GA implementation, see (Sheppard & Salzberg 1993). We show the results of the GA experiments in Figure 2. As with NN, the GA performs very well when evading one missile. In fact, it is able to achieve near perfect performance after 15,000 engagements and very good performance (above 90%) after only 5,000 engagements. Note that the number of engagements is somewhat inflated for the GA because it evaluates 50 plans during each generation. A generation is defined to be a stage in which the system evaluates each plan and then applies the genetic operators. In fact, the simulation ran for only 500 generations (i.e., 25,000 engagements) in these experiments.

The most striking difference in performance between NN and the genetic algorithm is that the GA learned excellent strategies for the two-missile problem, while nearest neighbor did not. Indeed, the GA achieved above 90% evasion after 16,000 engagements (320 generations) and continued to improve until it exceeded 95% evasion. This led to our idea that the GA could provide a good source of examples for NN. Thus, the GA became a "teacher" for NN.

Bootstrapping Nearest Neighbor

The idea is to use a GA to generate correctly labeled examples for the NN algorithm. This "teaching" should allow NN to take good actions at every time step, which hopefully will improve its success rate from the abysmal 45% it demonstrated previously on the two-missile problem. Teaching proceeds as follows. First, the GA is trained until it reaches a performance threshold, θ. From that point on, the system monitors the engagements used to test the GA. Any engagement that ends in success provides 20 examples (one for each time step) for NN. After 100 test engagements have been run through the GA in this manner, NN is tested (to estimate its performance) with an additional 100 random engagements. The examples continue to accumulate as the genetic algorithm learns the task.

The results of training NN using GA as the teacher (GANN) are shown in Figure 3. The figure shows the results of averaging over 10 trials, and it reflects experiments for three separate values of θ. The first threshold was set to 0%, which meant that all generations of the GA were used to teach NN. The second threshold was set to 50% to permit GA to achieve a level of success approximately equal to the best performance of NN on its own. Thus only generations achieving at least 50% evasion were used to produce examples for NN. Finally, the third threshold was set at 90% to limit examples for NN to extremely good experiences from the GA.

When $\theta = 0\%$, GANN starts performing at a level approximately equal to the best performance of NN. From there, behavior is erratic but steadily improves until ultimately reaching a performance of approximately 97% evasion. If we cut off the learning curve

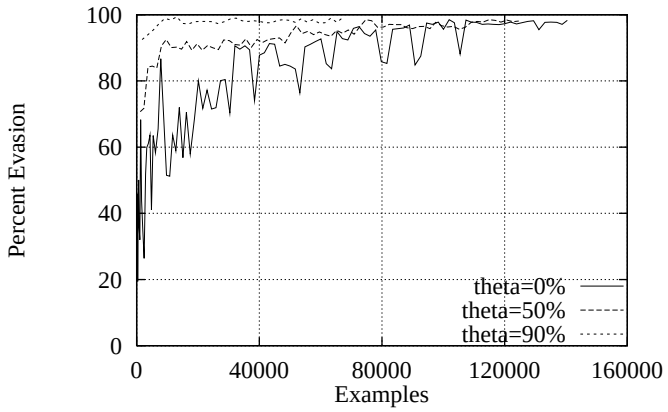

Figure 3: Results of nearest neighbor evasion using examples from the genetic algorithm with $\theta = 0\%$, $\theta = 50\%$, and $\theta = 90\%$.

after 50,000 examples (which is consistent with the NN experiments), performance still approaches 90%, but the overall behavior is still unstable. Nevertheless, we are already seeing substantial improvement in NN's performance on this task.

When $\theta = 50\%$, GANN starts performing at a very high level (above 70%) and quickly exceeds 90% evasion. In addition, the learning curve is much smoother, indicating more stability in the actions provided by the examples. Again, cutting the learning curve off at 50,000 examples, GANN is performing above 95% evasion, and some individual trials are achieving 100% evasion.

Finally, when $\theta = 90\%$, GANN started with excellent performance, exceeding 90% evasion with the first set of examples. GANN converged to near-perfect performance with only 10,000 examples. In fact, one trial achieved perfect performance with the first set of examples and remained at 100% evasion throughout the experiment. Another striking observation was that GANN was able to perform better than the GA throughout its learning. For example, when $\theta = 0\%$, GANN was achieving 50–80% evasion while the GA was still only achieving 2–10% evasion. Further, GANN remained ahead of the GA throughout training. Even when $\theta = 90\%$, GANN was able to achieve 98–100% evasion while the GA was still only achieving around 95% evasion. This indicated to us that we may be able to further reduce the number of examples and still perform extremely well.

Editing Nearest Neighbor

Our bootstrapping method showed that GANN can perform well with only a few examples from the genetic algorithm, and further that it can outperform its own teacher (the GA) during training. We decided to take our study one step further, and attempt to reduce the size of the example set without hurting performance. A large body of literature exists for editing example sets for nearest neighbor classifiers. Since NN is not usually applied to control tasks, though, we were not able to find any editing methods specifically tied to our type of problem. We therefore modified an existing editing algorithm for our problem. We call the resulting system GABED for GA Bootstrapping EDited nearest neighbor.

Early work by Wilson (Wilson 1972) showed that examples could be removed from a set used for classification, and that this simple editing could further improve classification accuracy. Wilson's algorithm was to use each point in the example set as a point to be classified and then classify the point with k-NN using the remaining examples. Those points that are incorrectly classified are deleted from the example set. Tomek (Tomek 1975) modified this approach by taking a sample of the examples and classifying them with the remaining examples. Editing then proceeds as in Wilson editing. Ritter *et al.* (Ritter *et al.* 1975) developed another editing method, which differs from Wilson in that points that are *correctly* classified are discarded. Wilson editing attempts to separate classification regions by removing ambiguous points, whereas the Ritter method attempts to define the boundaries between classes by eliminating points in the interior of the regions.

The editing approach we took combined the editing procedure of Ritter *et al.* and the sampling idea of Tomek. We began by selecting the example set with the fewest number of examples yielding 100% evasion. This set contained 1,700 examples. Next we edited the examples by classifying each point using the remaining points in the set. If a point was correctly classified, we deleted it with probability 0.25. (This probability was selected arbitrarily and was only used to show the progression of performance as editing occurred.) Prior to editing and after each pass through the data, the example set was tested using NN on 10,000 random engagements. During editing, classification was done using k-NN with $k = 5$.

The result of running GABED on the 1,700 examples is shown in Figure 4. Note that a logarithmic scale is used on the x-axis, because by editing examples with a 25% probability, more examples will be removed early in the process than later. Further, the graph shows "improvement" as the number of examples increases. Considered in reverse, it is significant to note that performance remains at a high level (greater than 90% evasion) with only 50 examples. And even with as few as 10 examples, GABED is achieving better than 80% evasion, which is substantially better than the best ever achieved by NN alone.

Discussion and Conclusions

The experiments reported here show that it is now possible to build efficient memory-based representations for delayed reinforcement problems. These experiments also demonstrate clearly the power of hav-

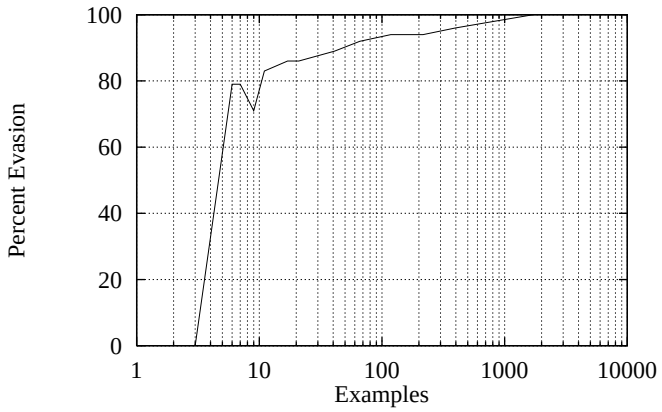

Figure 4: Results of editing examples provided by the genetic algorithm for k-nn.

ing a teacher or other source of good examples for memory-based methods when applied to complex control tasks. Without a reliable source of good examples, our memory-based method (k-NN) was unable to solve the problem, but with the good examples, it performed as well or better than the best of the other methods. In addition, we found that editing the example set can lead to a relatively small set of examples that do an excellent job at this complex task. It might be possible with careful editing to reduce the size of memory even further. This question is related to theoretical work by Salzberg *et al.* (Salzberg *et al.* 1991) that studies the question of how to find a minimal-size training set through the use of a "helpful teacher", which explicitly provides very good examples.

We note that when nearest neighbor began, its performance exceeded that of its teacher (the genetic algorithm). This indicates that perhaps the memory-based method could have been used at this point to teach the GA. We envision an architecture in which different learning algorithms take turns learning, depending on which one is learning most effectively at any given time. Such an architecture could lead to much faster training times.

This research demonstrates the potential for excellent performance of memory-based learning in reactive control when coupled with a learning teacher. We expect the general idea of using one algorithm to bootstrap or teach another would apply in many domains.

Acknowledgements

We wish to thank David Aha, John Grefenstette, Diana Gordon, and Sreerama Murthy for several helpful comments and ideas. This material is based upon work supported by the National Science foundation under Grant Nos. IRI-9116843 and IRI-9223591.

References

Aha, D., and Salzberg, S. 1993. Learning to catch: Applying nearest neighbor algorithms to dynamic control tasks. In *Proceedings of the Fourth International Workshop on AI and Statistics.*

Atkeson, C. 1989. Using local models to control movement. In *Neural Information Systems Conference.*

Barto, A.; Sutton, R.; and Anderson, C. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13:835–846.

Clouse, J., and Utgoff, P. 1992. A teaching method for reinforcement learning. In *Proceedings of the Machine Learning Conference.*

Grefenstette, J.; Ramsey, C.; and Schultz, A. 1990. Learning sequential decision rules using simulation models and competition. *Machine Learning* 5:355–381.

Grefenstette, J. 1991. Lamarckian learning in multiagent environments. In *Proceedings of the Fourth International Conference of Genetic Algorithms*, 303–310. Morgan Kaufmann.

Moore, A., and Atkeson, C. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

Ram, A., and Santamaria, J. C. 1993. Multistrategy learning in reactive control systems for autonomous robot navigation. *Informatica* 17(4):347–369.

Ram, A.; Arkin, R.; Boone, G.; and Pearce, M. 1994. Using genetic algorithms to learn reactive control parameters for autonomous robot navigation. *Adaptive Behavior* 2(3).

Ritter, G.; Woodruff, H.; Lowry, S.; and Isenhour, T. 1975. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory* 21(6):665–669.

Salzberg, S.; Delcher, A.; Heath, D.; and Kasif, S. 1991. Learning with a helpful teacher. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 705–711. Sydney, Australia: Morgan Kaufmann.

Sheppard, J., and Salzberg, S. 1993. Sequential decision making: An empirical analysis of three learning algorithms. Technical Report JHU-93/02, Dept. of Computer Science, Johns Hopkins University, Baltimore, Maryland.

Sutton, R. 1988. Learning to predict by methods of temporal differences. *Machine Learning* 3:9–44.

Tomek, I. 1975. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* 6(6):448–452.

Wilson, D. 1972. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics* 2(3):408–421.