Learning to Catch: Applying Nearest Neighbor Algorithms to Dynamic Control Tasks

Steven L. Salzberg and **David W. Aha**

Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218

Applied Physics Laboratory
The Johns Hopkins University
Laurel, MD 20723

Abstract

Models of dynamic control tasks are often inaccurate. Their accuracy can be improved through recalibration, which requires an enormous amount of data. An alternative approach improves a model by learning from experience; in particular, using nearest-neighbor and similar memory-based reasoning algorithms to improve performance. Recently these methods have begun to be explored by researchers studying dynamic control tasks, with some degree of success. For example, published demonstrations have shown how they can be used to simulate running machines, bat balls into a bucket, and balance poles on a moving cart. However, these demonstrations did not highlight the fact that small changes in these learning algorithms can dramatically alter their performance on dynamic control tasks. We describe several variations of these algorithms, and apply them to the problem of teaching a robot how to catch a baseball. We empirically investigate several hypotheses concerning design decisions that should be addressed when applying nearest neighbor algorithms to dynamic control tasks. Our results highlight several strengths and limitations of memory-based control methods.

Introduction

Dynamic control problems are the subject of much research in machine learning (e.g., Selfridge, Sutton, & Barto, 1985; Sammut, 1990; Sutton, 1990). Some of these studies investigated how to apply k-nearest neighbor methods to these tasks by modifying control strategies based on previously gained experience (e.g., Connell & Utgoff, 1987; Atkeson, 1989; Moore, 1990; 1991). However, these previous studies did not highlight the fact that small changes in the design of these algorithms drastically alter their learning behavior. This paper describes a preliminary study that investigates this issue in the context of a difficult dynamic control task: learning to catch a ball moving in a three-dimensional space, an important problem in robotics research (Geng *et al.*, 1991).

Our thesis in this paper is that entities can improve substantially at physical tasks by storing experiences without explicitly modeling the physics of the world. Although it is useful to exploit such models or use them in combination with stored experiences, we believe that people often use large numbers of stored experiences to guide their actions. For example, many athletes train by repeating the same actions thousands of times. That is, the skills we wish to study are gained through *practice*, during which we assume the learner stores experiences, which are retrieved for later use by relatively simple pattern-matching methods.

Our simulation-based experiments comprise an initial investigation of the algorithmic choices required when designing simple k-nearest neighbor algorithms that learn physical skills. In particular, we address the following questions:

1. What is the best representation to use?
2. Can nearest neighbor quickly learn rough approximations for control tasks?
3. Does k-NN outperform simple 1-NN, and does either outperform a simple non-learning control strategy?
4. What storage reduction strategies will allow the algorithms to save fewer instances without losing accuracy?

Related Work

The most frequently addressed learning task involving dynamic control is the cart-and-pole problem (Michie and Chambers, 1968), which requires balancing a pole standing on a cart. Learning involves selecting which of the two directions to push the cart along its one-dimensional track of fixed length. This task is difficult because it is not obvious which selection was responsible for causing a pole to fall since the system may have been in "trouble" for many time steps. Neural net (Selfridge, Sutton, & Barto, 1985) and nearest neighbor approaches (Connell & Utgoff, 1987) have performed reasonably well on this problem.

Another delayed-feedback problem is that of "juggling" a ball so that it falls repeatedly on a paddle held by a robot (Rizzi & Koditschek, 1988; Aboaf, Drucker, & Atkeson, 1989). Aboaf and his colleagues programmed their robot to learn how to calibrate its

motions. That is, the ball-juggling strategy was built into its software, but the vision system and the robot effectors introduced noise and calibration errors. Thus, the robot could not juggle the ball without substantial tuning. Therefore, they taught their robot how to improve its performance by adjusting its aim according to a polynomial fitted to the robot's errors.

We chose to investigate a variant of these control problems in which a simulated robot, allowed to move in any direction on a two-dimensional plane, mimics a center fielder attempting to catch a baseball flying at high speeds in a three-dimensional space. Like the other tasks, this problem involves predicting where an object will fall and positioning something at this location, although in this case the object to be positioned is the robot's glove rather than a cart or a paddle. However, the baseball-catching task is comparatively difficult; unlike the pole-balancing task, actions take place in a three- rather than a two-dimensional space, and, unlike the juggling task, the ball moves at high speeds, which complicates the catching task.

Geng *et al.* (1991) studied a similar catching task in which a stationary, robotic arm was programmed to catch a thrown ball. However, they did not investigate whether learning methods could be used to reduce the reliance on physical models and extensive calibration to solve their task.

Learning Objectives

In our simulated environment, a training trial begins with the robot standing in the center fielder's position in a baseball field. The simulation "bats" a ball from home plate in the robot's general direction. The trial continues until the ball lands or is caught. Simple physical equations of motion, incorporating gravitational acceleration but not friction or wind resistance, are used to determine the ball's location during each time step of its flight. We will use the term *flight* to indicate a complete trial from the time the ball is batted until the time it lands or is caught.

Instances are computed for each time step, are represented by simple attribute-value pairs, and always contain exactly two pieces of information: the current state of the ball-robot system, and the ball's landing location. The robot must learning a function mapping the current state to a recommended direction and distance for its own motion. Each instance encodes one element of this function (i.e., all instances are "positive" examples of the function). Only a small fraction of the possible states are encountered during training; thus, generating accurate predictions relies on correctly generalising from the stored instances.

The robot's objective is to catch the ball with a fixed-size glove. The robot is not given any model of physics, and it does not attempt to learn the equations governing the ball's flight path. Each time step consists of the robot consulting its previously stored instances in order to decide how to move. The robot moves at a constant speed, except when it is very close to the ball, when it is allowed to slow down. The simulation ensures that it is possible for the robot to catch every ball; the ball lands within a circle centered on the robot, and the robot is sufficiently fast to move to any location within this circle before the ball lands. Training the robot to move to the *correct* location within this circle is the performance objective of this task.

For our experiments, a time step lasted 0.1 seconds, and the initial distance between the ball and robot was 90 meters. The ball remained in the air for 4.5 seconds and always landed within 30 meters of the robot's starting position. The exact landing location was randomly selected. The robot can reach its arm and glove out 1 meter; i.e., it successfully catches a ball when it is within 1 meter of the ball's location.

We used a simulation rather than a real robot for two reasons:

1. The robot requires a large number of experiences before it reaches its peak performance. Running a real robot through tens of thousands of experiences is prohibitively time-consuming. Also, the robot's hardware may not easily withstand such drastic use.
2. To test our hypotheses, we wanted first to consider an environment in which sensor noise was not a problem. After finding the best setup for learning with noise-free data, we can then consider the effect of noise on the algorithm.

Nonetheless, we recognize the need to test our hypotheses and algorithms on robots rather than simulations. We plan to conduct such experiments in our future research.

Learning Algorithms

The algorithms we investigated in our experiments for learning the ball-catching task are all variants of *k*-nearest neighbor methods, which derive predictions for a newly presented instance based on a set of similar stored instances (Dasarathy, 1990). This family of algorithms has been awarded several names in the machine learning and related literatures, including *instance-based* learning (Bradshaw, 1987; Aha, Kibler, & Albert, 1991), *exemplar-based* learning (Salzberg, 1991), and *memory-based* reasoning/learning (Stanfill & Waltz, 1986; Atkeson, 1989; Moore, 1990). We include in this family of algorithms more sophisticated methods that first extract the nearest neighbors, and then attempt relatively expensive function estimation methods using those neighbors (e.g., local weighted regression).

Initially, the robot has no stored instances to use for predicting which direction to move. During the first training flight, it uses the simple default rule "move towards the ball." One instance is computed for each time step during each flight, but instances are not stored until the flight completes.

All stored instances include information on which direction the robot should move when in the instance's state (i.e., this information is obtained by comparing the landing location of their flight's ball with their robot's location). For subsequent flights, the learning algorithm proceeds by computing similarities between the current state and the states of all stored instances, selecting the k most similar instances, and using a function of their movement directions to determine which direction to move the robot. Distance is simply Euclidean distance in the attribute space.

Predictions and Hypotheses

The selection of the state representation is crucial for simple k-nearest neighbor algorithms. If the state's dimensions are relatively uninformative, due to the fact that they do not encode important relationships between the ball's flight and the robot's location, then no learning algorithm can be expected to have fast learning rates. Furthermore, since k-nearest neighbor algorithms are not robust in the presence of irrelevant attributes (Aha, 1989; Kelly & Davis, 1991), it is important to minimise the number of irrelevant attribute dimensions used to represent the state. Our algorithms should have faster learning rates when the state representations contain informative and relevant attributes. This claim is examined in the experiments section.

Only after a good state representation has been chosen can we determine which of several possible variants of our algorithms has the best learning bias for the ball-catching task. We varied a number of parameters in our simulation in an attempt to find the best algorithm for the task. The following list summarises our expectations:

1. *Similarity*: Domain-specific scaling of specific attributes allows more accurate similarities to be computed, which should increase learning rates.
2. *Prediction*: Higher values of k (i.e., considering more neighbors) will improve performance. Also, different methods for combining $k > 1$ directions may change learning rates.
3. *Learning*: Storage reduction strategies can be used to significantly decrease the number of instances required by the robot to catch the ball with high frequency.

Experiments

We conducted learning trials with several different k-nearest neighbor algorithms. We define a trial to be a sequence of training flights, each one independently generated, with periodic testing. For a given trial, the same set of test flights was used for every algorithm.

All experiments, except where noted otherwise, shared the following features:

1. the results were averaged over ten trials,
2. a training trial consisted of 500 randomly generated flights,
3. accuracy was measured on an independent set of 100 test flights, and
4. attribute-values were linearly normalized to the range [-1,1].

The dependent variables measured in all of our experiments are: (1) the distance of the robot from the ball's landing location, (2)the percentage of the balls "caught"; i.e., less than one meter from the robot when they landed, and (3) storage requirements. The distance of the robot from the ball measures how close the learned function is to a perfect control strategy. However, attaining perfect accuracy for dynamic control tasks with real-valued attributes is impossible, at least to this degree of detail. Thus, the percentage of balls caught is a useful measure of how the robot is performing; this tests how frequently the robot comes very close to the ball's landing location.

The independent variables considered in our experiments are:

1. the representation used to encode states and instances,
2. the number k of most similar neighbors used to make k-NN predictions,
3. the method by which the prediction of the neighbors are combined, when $k > 1$, and
4. the storage reduction strategy.

The values used for the independent variables and the results of our experiments are described in the following sections.

Varying the Representation for States

We varied the representation used for states to determine which, among a small set, would provide a good representation for k-nearest neighbor learning. We compared three different representations for states, which we will call **complete**, **irrelevant**, and **useful** due to their characteristics. The first of these representations has eight dimensions: the ball's three-dimensional location, its three-dimensional velocity, and the robot's two-dimensional location. This representation is complete but uninformative, because each instance it represents contributes a relatively small amount of information. One problem is that it encodes a third-party view of the task rather than a robot-centered view. Thus, when the robot finds itself in a position similar to one it had previously encountered (i.e., one in which the relative positions of the ball and robot are similar to those of a stored instance), the stored instance's state may appear to be quite different, because the absolute locations of the ball and robot may be very different. That is, there is a many-to-one mapping of instances in this representation to instances in a robot-centered representation.

The second representation, **robot-centered**, considers attributes from the point of view of the robot. It replaces four attributes, the X and Y positions of the

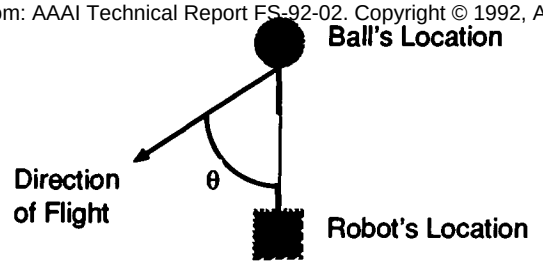

Figure 1: The angle θ marks one of the attributes used in the **robot-centered** and **useful** state representations tested in the experiments summarized in Table 1.

Table 1: 1-NN's performance using different representations.

Representation	Distance	% Caught
Complete	21.3	0.0
Robot-centered	1.5	38.3
Useful	0.5	92.7

ball and robot, with two attributes, the **ball-robot distance** and the angle θ of the ball's flight direction with respect to the robot's location, as shown in Figure 1.[1] Using this representation, many seemingly different absolute ball and robot locations having similar relative ball-robot relationships are encoded similarly.

However, in the robot-centered representation two of the attributes are still somewhat irrelevant: the X and Y and velocities of the ball during a flight are constant within a flight and almost identical among different flights in our experiments. These attributes thus provide little information for future movement decisions. Our third representation, called **useful**, is just the **robot-centered** representation with these two attributes are discarded.

Not surprisingly, the **useful** representation supports the fastest learning rates. Table 1 summarizes the results when the 1-nearest neighbor algorithm is used to control movement direction. The **useful** representation was used for all our subsequent experiments.

Simpler control strategies

We also considered the possibility that a simple non-learning heuristic might lead to reasonably good performance at our task, so we conducted an experiment to test the following strategy. The robot simply followed the ball's ⟨x,y⟩ position at all times. We hypothesized that, in our domain, the robot would perform badly with this default strategy because it moves much more slowly than the ball. With this strategy, the robot would initially run towards the ball, which would eventually be directly overhead. The robot is not fast enough to turn around and catch up to the ball once it passes overhead.

Our experiments showed that this is exactly what happened. Using the **useful** representation, the robot's performance is significantly better using 1-NN than the simple "follow-the-ball" default control strategy, whose average distance between robot and ball-landing location was 18.84 meters, and it caught none of the balls from the test flights even after processing 500 training flights.

A graphic example

Figure 2 provides some intuition about the robot's movements. The figure displays the flight of the ball, commencing from home plate at the top of the graphs, and the movements chosen by the robot using the basic 1-NN algorithm with the **useful** representation. The upper left graph demonstrates that, on the first training instance, the robot simply moves toward the ball's (X,Y) position. Thus, the ball eventually flies over the robot's head and lands far away from the robot's final position (i.e., 6.14 meters). The other three graphs show where the robot moves on the same test flight after 1, 10, and 100 training flights have been processed. After only one training flight, the ball naturally moves in the wrong direction (i.e., towards the right) since all similar instances derived from the first training flight tell the robot to move to the right, and then move away from home plate towards the end of the ball's flight. In this case, the ball landed 37.84 meters from the robot. After processing 10 training flights, the robot has a much richer set of instances to draw from and it moves much closer to its goal, getting within 3.69 meters of the ball's landing location. Finally, the lower-right graph shows how the robot behaves after training on 100 flights; it catches the test flight's ball, having moved within 0.35 meters of the ball's landing location.

Varying the Similarity Function

While linear normalizations of values are typically used for *k*-nearest neighbor algorithms to ensure that each attribute has equal weight in the similarity function, scaling different dimensions differently helps to tolerate irrelevant attributes and/or increase the attention given to relevant attributes. This has often been demonstrated by tuning explicit attribute weights (e.g., Stanfill & Waltz, 1986; Aha, 1989; Salzberg, 1991; Kelly & Davis, 1991). An alternative scaling method is to *normalize* different attributes differently to ensure that the difference between an attribute's values is deemed more significant for one subrange of its values than for other subranges.

In our simulations, we found that the robot quickly learns how to get within a few feet of the ball's landing

[1] Left-bearing angles were defined to be negative and right-bearing angles were defined to be positive. Angles ranged in value between π and π radians. Care was taken to ensure that the similarity function understood that large negative angles were similar to large positive angles; that values of $-\pi$ and π were to be treated as identical for this attribute.

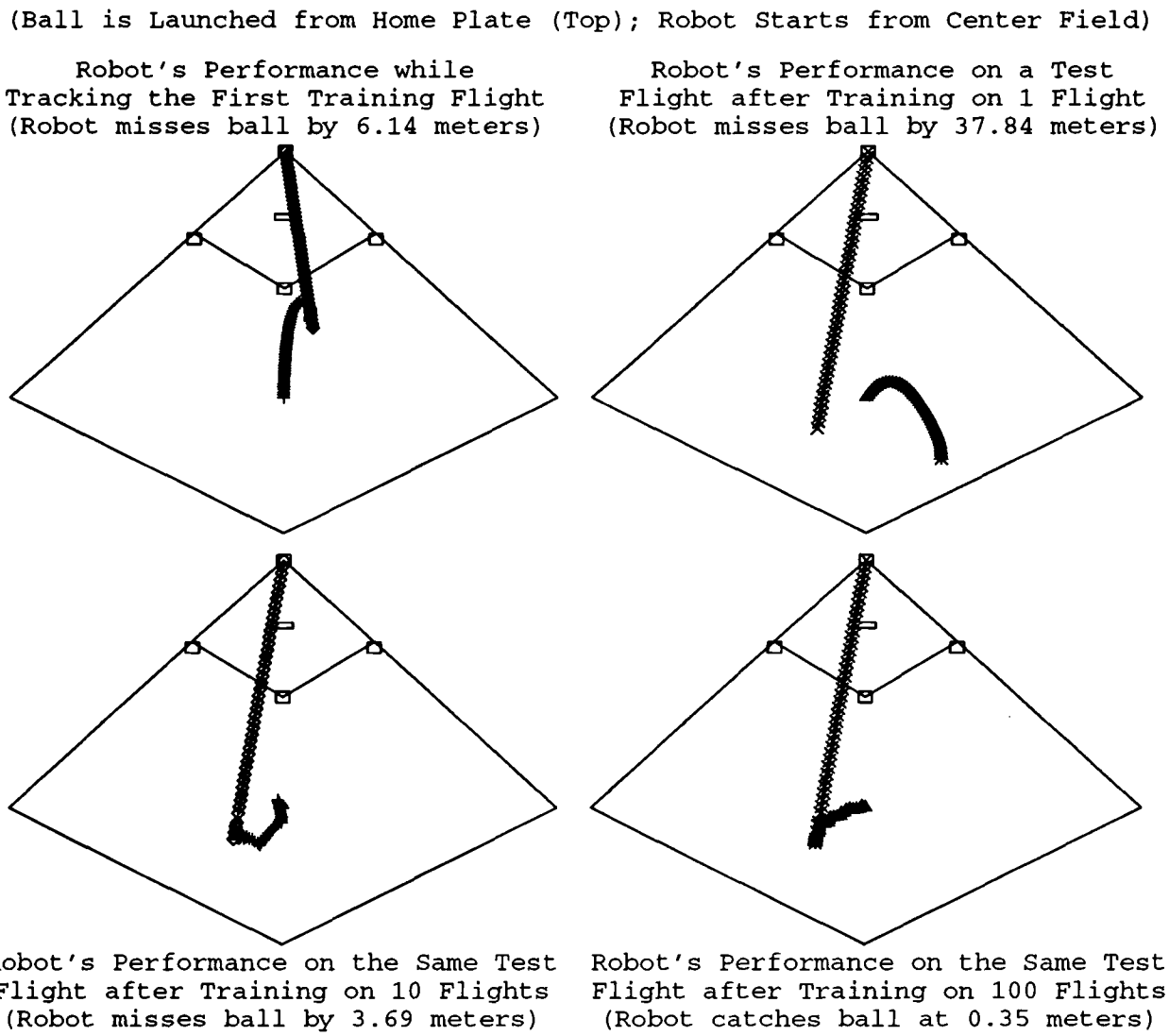

Figure 2: These graphs show where the robot moves as it follows the ball during the first training flight (upper left) and as it tracks the same test flight, using 1-NN, after different numbers of training flights.

location, but requires far more training flights to catch it with regularity. An examination revealed that the **ball-robot distance** attribute was not being given sufficient weight in the similarity function; small differences in this value among new and stored instances were ignored because their normalized values differed by only a small amount whenever the robot was "close to" the ball's ⟨X,Y⟩ position. Since this was true for many robot-ball positions, and since the **angle** attribute continued to vary greatly between similar such robot-ball positions, the similarity function focused on matching the **angle** values in these circumstances, neglecting to consider whether the stored instance's **ball-robot distance** closely matched the current state's value.

One way to force the similarity function to give more attention to **ball-robot distance** is to weight that more highly in the similarity function. However, this is useful only when the robot is close to the ball. Instead of using an instance-specific weighting strategy (Aha & Goldstone, 1992) to solve this problem, we applied a non-linear normalization function to **ball-robot distance** values, which forced them to be more different for small values.[2] We empirically compared the robot's performance when using a linear versus an exponential distribution on **ball-robot distance**. Performance was clearly better when using the exponential distribution. Figure 3 summarizes the cases for 1-NN and 5-NN. 5-NN used the unweighted average of the 5 stored directions to determine which direction the robot should move. Learning rate improvements caused by using the exponential normalization function for **ball-robot distance** were even more drastic when using it in combination with our storage-reduction strategies.

These curves show that simple k-nearest neighbor algorithms can quickly learn reasonably accurate functions for this task. For example, the robot caught an average of over 90% of the balls in the test flights after training on only 100 training flights when using 1-NN with an exponential normalization for **ball-robot distance**. Thus, it can quickly produce roughly accurate behavior. However, the learning curve here also shows that many more training flights are required before the robot approaches perfect accuracy. Better representations and scaling methods might increase these algorithms' learning rates.

Varying the Prediction Function

It is well-known that higher values for k generally increase learning rates (e.g., Dasarathy, 1990). We expected this to be true for the ball-catching task when a good representation for states is used. Likewise, different methods for combining $k > 1$ movement directions should affect learning rates.

[2] The exponential normalization function was $f(\text{distance}) = 1 - e^{-0.05 \times \text{distance}}$.

Figure 3 displays evidence that 5-NN outperforms 1-NN on the ball-catching task when using the **useful** representation for states. We also conducted experiments comparing unweighted 5-NN to a simple similarity-weighted variant of 5-NN. In the similarity-weighted algorithm, each of the five most similar stored instances has a different effect on the direction the robot moves, with the effect decreasing linearly with distance from the new instance. This algorithm actually decreased accuracy in our experiments. For example, average landing distance after training on 500 flights increases from 0.40 to 0.59 meters and the percentage of balls caught decreases from 98.0 to 89.9, when using the similarity-weighted 5-NN algorithm with a linear normalization function. However, other functions of the $k > 1$ most similar instances' predictions have been shown to increase learning rates on other applications (e.g., Atkeson, 1989). We are currently experimenting with kernal regression, localized weighted regression, and other such prediction functions in an effort to determine how fast other k-NN variants can learn this task.

Varying the Learning Function

The previous algorithms saved every instance encountered during all training flights. We would like to minimise the number of stored instances and still maintain accurate performance. While it is well-known that many different storage reduction strategies for k-NN algorithms can be usefully employed without significantly decreasing performance on some learning tasks (e.g., Hart, 1967; Connell & Utgoff, 1987; Moore, 1990; Aha, Kibler, & Albert, 1991; Salzberg, 1991; Heath *et al.*, 1991), it is not obvious which approach is best for this problem nor to what degree learning rates are effected. Nonetheless, we hypothesized that significant storage reductions could be obtained by simple storage-reduction strategies without significantly reducing the robot's performance.

We examined three storage-reduction strategies:

1. **None**: All instances are stored.
2. **Close-Only**: Store only those instances whose normalized **ball-robot distance** is less than a fixed threshold value.
3. **Cover**: Store only those instances whose normalized distance to their nearest neighbor is greater than a fixed threshold value.

The motivation for the **Close-Only** strategy was to investigate whether saving "close" instances only would suffice for catching the ball with high frequency. The motivation for studying the **Cover** strategy, first used by Sebestyen (1962), was to sample the instance space broadly, but not to save instances in areas of the space which had already been sampled. Since the environment is noise-free, such instances presumably do not provide much additional information.

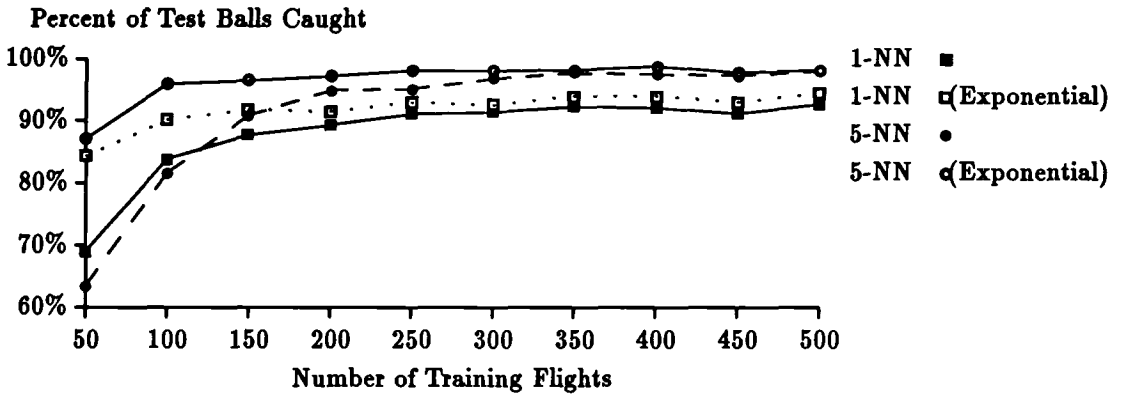

Figure 3: The robot learns more quickly when using an exponential normalization function for the ball-robot distance attribute.

Table 2: Comparative results from using two storage-reduction strategies.

Storage Strategy	k	Distance (meters)	% Caught	Storage
None	1	0.5	94.4	22,000
	5	0.4	98.2	22,000
Close-Only	1	11.8	29.3	6,021
	5	10.6	34.9	6,284
Cover	1	0.5	89.8	2,951
	5	0.4	97.7	2,518

Table 2 summarizes the results for these three storage reduction strategies, where the threshold for **Close-Only** was 45, meaning only instances less than 45 meters from the robot would be stored. This table lists the storage used (i.e., the number of instances stored) after training on 500 flights. As usual, accuracy was measured on an independent test set of 100 flights and results are averaged over 10 trials. When using the **Close-Only** storage-reduction strategy, the robot performed poorly. Apparently, "distant" instances are indeed used to help select good directions for the robot to move while tracking the ball. However, the robot's performance while using the **Cover** strategy reduced storage requirements by an order of magnitude while sacrificing accuracy only by a small amount — and with virtually no sacrifice for 5-NN. Thus, the **Cover** storage reduction strategy works well for this domain.

Discussion

The goal of our learning algorithms is *not* to induce the rules of physics governing the ball's flight, but rather to learn a mapping from states describing the ball's relative position and velocity to an action that the robot should take. We applied variants of k-nearest neighbor algorithms (k-NN) to solve this ball-catching task. Several researchers have recently demonstrated how to use similar algorithms to solve other dynamic control tasks (e.g., Connell & Utgoff, 1987; Atkeson & Reinkensmeyer, 1989; Moore, 1990). Our results show that nearest neighbor methods provide a good means for estimating roughly where a ball will land. When the ball is far away, our algorithm usually guides the robot in roughly the correct direction. However, when the robot is near the landing location of the ball, it has difficulty moving correctly. This is perhaps the most interesting result from our study. The difficulty is due to the fact that the function the robot is learning actually has a point with infinite derivative, and any approximation method will have trouble making correct predictions near this point in attribute space. This problem area occurs where the robot is directly in the ball's flight path, and the ball is going to land very close to the robot. If the ball is going to land any distance $\epsilon > 0$ in front of the robot, the correct direction to move is straight ahead; however, if the ball will land some distance ϵ behind the robot, the correct direction to move is just the opposite. The intuitive interpretation of these results is that it is easy to learn approximately where a ball (or other projectile) will land, even when using simple 1-NN methods. However, more elaborate control algorithms may be required to allow the robot move more precisely to where the ball will land.

Conclusions and Next Steps

To summarize, our experiments with this domain suggest that:

1. instance spaces defined by informative and relevant attributes improves performance for k-nearest neighbor methods,
2. non-linear scaling methods can significantly increase learning rates by improving the accuracy of similar-

ity computations,

3. increasing k can increase learning rates, and
4. storage reduction strategies can reduce storage by an order of magnitude without substantially sacrificing accuracy.

Our next step will be to improve the physics of the simulation and relax the constraints of the problem. For example, we plan to remove the idealized assumptions about the world and introduce sensor noise, air friction, wind, effector noise (i.e., the robot's actual motions will not be identical to its intended motions) and varied flight times to examine their effects on learning performance.[3] More generally, we are interested in how well k-nearest neighbor performance generalises to different problems. Finally, the playing field itself will be changed through the addition of barriers and other complications.

Acknowledgements

Thanks to Simon Kasif and John Sheppard for discussions and pointers concerning this project. This work was supported in part by the National Science Foundation under Grant IRI-9116843.

References

Aboaf, E., Drucker, S., & Atkeson, C. (1989). Task-level robot learning: Juggling a tennis ball more accurately. In *Proceedings of the IEEE Internatl. Conf. on Robotics and Automation*. IEEE Press.

Aha, D. W. (1989). Incremental, instance-based learning of independent and graded concept descriptions. In *Proc. Sixth Internatl. Wkshop. on Mach. Learning* (pp. 387–391). Ithaca, NY: Morgan Kaufmann.

Aha, D. W., & Goldstone, R. L. (1992). Concept learning and flexible weighting. To appear in *Proc. 14th Annual Conf. Cognitive Sci. Soc.*. Bloomington, IN: Lawrence Erlbaum.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Atkeson, C. (1989). Using local models to control movement. *Proc. Neural Info. Processing Systems.*

Atkeson, C., & Reinkensmeyer, D. (1989). Using associative content-addressable memories to control robots. In *Proc. Internl. Conf. on Robotics and Automation*. IEEE Press.

Bradshaw, G. (1987). Learning about speech sounds: The NEXUS project. In *Proc. Fourth Internatl. Wkshop. on Mach. Learning* (pp. 1–11). Irvine, CA: Morgan Kaufmann

Connell, M. E., & Utgoff, P. E. (1987). Learning to control a dynamic physical system. In *Proc. Sixth Natl. Conf. on Artificial Intelligence* (pp. 456–460). Seattle, WA: Morgan Kaufmann.

[3] A preliminary study showed that, when allowing the ball to remain in flight for between 4.0 and 5.0 seconds rather than fixing its time at 4.5 seconds, 1-NN accuracy dropped from 0.5 to 0.6 meters in distance and 92.7% to 84.9% balls caught.

Dasarathy, B. V. (Ed.). (1990). *Nearest neighbor(NN) norms: NN pattern classification techniques*. Los Alamitos, CA: IEEE Computer Society Press.

Geng, Z., Fiala, J., Haynes, L. S., Bukowski, R., Santucci, A., & Coleman, N. (1991). Robotic hand-eye coordination. In *Proc. 4th World Conf. on Robotics Research*. Pittsburgh, PA.

Hart, P. E. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory, 14*, 515–516.

Heath, D, Kasif, S., Kosaraju, R., Salzberg, S., & Sullivan, G. (1991) Learning nested concept classes with limited storage. In *Proc. Twelfth Internatl. Joint Conf. on Artificial Intelligence* (pp. 777-782). Sydney, Australia: Morgan Kaufmann.

Kelly, J. D., Jr., & Davis, L. (1991). A hybrid genetic algorithm for classification. In *Proc. Twelfth Internatl. Joint Conf. on Artificial Intelligence* (pp. 645–650). Sydney, Australia: Morgan Kaufmann.

Michie, D., & Chambers, R. (1968). Boxes: An experiment in adaptive control. In E. Dale & D. Michie (Eds.), *Machine intelligence 2*. Edinburgh, Scotland: Oliver and Boyd.

Moore, A. W. (1990). Acquisition of dynamic control knowledge for a robotic manipulator. In *Proc. Seventh Internatl. Conf. on Mach. Learning* (pp. 244–252). Austin, TX: Morgan Kaufmann.

Moore, A. W. (1991). Variable-resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proc. Eighth Internatl. Mach. Learning Wkshop.* (pp. 333–337). Evanston, IL: Morgan Kaufmann.

Rizzi, A., Whitcomb, L., & Koditschek, D. (1991). Distributed real-time control of a spatial robot juggler. *IEEE Computer*, 1991.

Salzberg, S. L. (1991). A nearest hyperrectangle learning method. *Machine Learning, 6*, 251–276.

Sammut, C. (1990). Is learning rate a good performance criterion for learning? In *Proc. Seventh Internatl. Conf. on Mach. Learning* (pp. 170–178). Austin, TX: Morgan Kaufmann.

Sebestyen, G. S. (1962). *Decision-making processes in pattern recognition*. New York, NY: Macmillan.

Selfridge, O. G., Sutton, R. S., & Barto, A. G. (1985). Training and tracking in robotics. In *Proc. Ninth Internatl. Joint Conf. on Artificial Intelligence* (pp. 670–672). Los Angeles, CA: Morgan Kaufmann.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM, 29*, 1213–1228.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. Seventh Internatl. Conf. on Mach. Learning* (pp. 216–224). Austin, TX: Morgan Kaufmann.