Iterated Phantom Induction: A Knowledge-Based Approach to Learning Control

MARK BRODIE*

mbrodie@us.ibm.com

IBM TJ Watson Research Center, Hawthorne, New York, USA

GERALD DEJONG

dejong@cs.uiuc.edu

*Department of Computer Science, Beckman Institute, University of Illinois at Urbana-Champaign,
Illinois, USA*

Editor: William W. Cohen

Abstract. We advance a knowledge-based learning method that allows prior domain knowledge to be effectively utilized by machine learning systems. The domain knowledge is incorporated not into the learning algorithm itself but instead affects only the training data. The domain knowledge is used to explain and then transform the actual training examples into a more informative set of imaginary, or “phantom” examples. These phantom examples are added to the training set; the experienced examples are discarded. A new control policy is induced from the phantom training set. This policy is then exercised, yielding additional training points, and the process repeats.

We investigate the performance of this method in a stylized air-hockey domain which demands a difficult nonlinear control policy. Our experiments show that, surprisingly, an accurate policy can be learned even if the domain theory is only imprecise and approximate. We advance an interpretation which indicates that the information available from a plausible qualitative domain theory is sufficient for robust successful learning. This interpretation is used to make a number of predictions which are tested in subsequent experiments. The outcomes confirm the interpretation and the robustness of the approach.

Keywords: control learning, explanation-based learning, reinforcement learning, prior knowledge, phantom points

1. Introduction

1.1. Prior knowledge

The use of prior knowledge has a long history in Artificial Intelligence. In machine learning the best results are usually obtained by systems that make use of specialized domain knowledge. So-called “weak” methods—general methods that apply to a wide variety of domains—tend to be out-performed by strong methods which reflect specific adjustment or tailoring to a given task (Rich & Knight, 1991; Russell & Norvig, 1995). This is not surprising, because knowledge focuses attention and reduces search time. The use of domain knowledge allows learning to converge more quickly because one piece of information can substitute for many training examples. It is wasteful to consume training examples in order to learn things which are already known. It is more efficient for the system to be given

*Work done at the Department of Computer Science of the University of Illinois at Urbana-Champaign.

prior knowledge about the domain and to exploit the training examples to learn what is not known.

Incorporating domain knowledge requires that certain challenges be addressed. Firstly, the knowledge possessed by human domain experts is usually difficult to express in machine learning terms. In order to be effectively exploited, the expert's prior knowledge must first be transformed into the internal vocabulary of a particular learning algorithm: e.g. the weights of a neural network, the utility function of a genetic algorithm, and so on. These representations are seldom amenable to easily encoding prior knowledge. This is why incorporating prior knowledge is usually done in a somewhat ad-hoc fashion which differs from domain to domain. For complex, real-world problems the most effective way of taking advantage of prior knowledge is to design a learning algorithm with that specific knowledge in mind.

If the knowledge base or the domain of application changes, the difficult process of incorporating prior knowledge has to be repeated. Thus methods that effectively utilize domain knowledge are not easily modified from their original domain of application and its specific knowledge base. Greater generality would be achieved if human experts could state their knowledge naturally in a language appropriate to the domain, with the machine learning system shouldering the responsibility of making use of the knowledge without any additional programming being required.

A second difficulty with utilizing prior knowledge is that the resulting systems can be brittle. A system that is heavily reliant on prior knowledge may fail catastrophically if that knowledge is inaccurate or incomplete. This occurs because the domain knowledge is unable to account for all the possible eventualities that the system may encounter (the well-known "qualification problem" (McCarthy & Hayes, 1969; Genesereth & Nilsson, 1987)). To alleviate this difficulty the domain theory needs to be sensitive to the actual behavior exhibited by the real world, so that the system can respond appropriately in situations where its knowledge is inadequate.

In this work we propose a solution to these two difficulties. We suggest a general approach which allows naturally expressed human domain knowledge to be effectively utilized by machine learning systems. At the same time brittleness can be avoided by inferentially refining the domain knowledge through interactions with the real world. Thus the system can respond flexibly by utilizing both its prior knowledge and the feedback it receives from the world. The most significant aspect of this research is that the generalization method need not be specifically designed or adapted to the prior knowledge.

1.2. Learning control

Our approach is applicable to learning to perform control tasks. This is an important class of problems which require adapting to the environment in order to learn an adequate control policy. Human experts often possess prior knowledge about such problems, and we would like our machine learning systems to be able to exploit this knowledge. However the system also needs to be able to respond to unforeseen eventualities, where its prior domain theory may be un-informative. The information obtained from interacting with the world must be combined with the prior domain knowledge to achieve good performance quickly.

Figure 1. Interactive learning.

We adopt a framework of *interactive learning* (see figure 1). The system maintains a *policy* which represents its opinion as to how to perform the tasks it faces. Problems are successively given to the system. For each problem a solution is generated according to the system's current policy. This hypothesized solution is tried out in the world and the result is observed. Periodically, after accumulating such examples of world behavior, the learning module may revise its policy (see figure 1).

More specifically, let S be the set of world states, D the set of decisions the system can take, and F the family of policies available to the system; each $f \in F$ is a mapping from S into D . We assume that there is some optimal policy f^* which computes the best decision for each state. However it is unrealistic to assume we know the form of f^* , thus we do not assume $f^* \in F$. The learning task is to find the element in F that is closest to f^* —this will be called the **target policy**.

The system maintains a current policy $f_i \in F$. Learning proceeds by observing the effects of policy f_i in the world and using this feedback to update the policy to f_{i+1} . Clearly, if one wishes to take advantage of the prior knowledge that is available, the algorithm that is used to generate policy f_{i+1} could be explicitly designed to exploit this knowledge. However, as explained in the introduction, this method is expensive and lacks generality. The next section outlines a different approach.

1.3. "Phantom" training

We call our approach "*phantom induction*" (Brodie & DeJong, 1998). The key idea is that the training set does not consist of the actual experiences of the system. Instead, each observed example is transformed using prior domain knowledge into an imaginary, or "phantom," experience. These phantom experiences are based on both the information contained in the observed examples and also the information obtained from the prior knowledge. They can therefore be more useful as training examples than the actual observed experiences. The important point is that the algorithm used to generalize from the training examples need have no connection whatsoever with the prior knowledge. Figure 2 illustrates this point by showing an expanded version of figure 1's learning module for the case of phantom induction.

More specifically, the method proceeds as follows: Each time the system makes a decision, it observes the resulting behavior in the world. The *observed error* is the difference between

Figure 2. The learning module for phantom induction. The domain knowledge does not affect the algorithm used to generate new policies.

the desired goal and the actual outcome. Domain knowledge is used to explain the observed world behavior (DeJong, 1994). The observed error is propagated through the explanation, resulting in a characterization of the *decision error* for the actual decision. From this decision error and the actual decision a “phantom decision” d_{ph} is produced. According to the system’s model of the world, choosing this phantom decision in place of the original decision would have yielded an adequate solution to the problem.

Each phantom decision is a hypothetical point which the system has not actually experienced. If the domain theory and, therefore, the explanation were perfect, the phantom decision would be successful; i.e. it would lie along the graph of the optimal policy f^* , constituting an actual positive training example. If instead the system’s domain knowledge is flawed, the phantom decision may lie far from the optimal decision. Our suggestion is to nonetheless treat each phantom decision as if it were an actually experienced positive training example (i.e. as if it were the decision that would achieve the goal for that state). Using an inductive algorithm, a new policy is generated from the collection of phantom observations. The process is iterated: the system attempts to solve further problems using its new policy; from the observed world behavior new phantom decisions are generated; and finally, these are added to the collection of phantom observations, which are used to generate another policy. We should emphasize that the actual decisions are not added to the training set; the training examples consist entirely of phantom decisions. However if the actual decision achieves the goal, then the decision error is zero and the phantom decision coincides with the actual decision.

The domain knowledge is used to perturb the observed training examples into phantom training examples, without requiring any commitment to a particular inductive algorithm. It is intuitively clear that the performance of this iterated phantom induction process will depend on the accuracy of the explanations. If the explanations are in some sense “nearly” correct then the phantom decisions should provide more information about the correct decisions than the actual decisions do. In this case we might expect the decision policy to converge to an adequate policy.

We conducted a sequence of experiments which uncovered some interesting properties:

- (1) The method can be successful even with highly imprecise explanatory information.

(2) Learning requires only a tractable number of examples. (3) Convergence is not monotonic: intermediate strategies can perform dramatically worse than earlier strategies and yet these poorly-performing strategies must be exercised for the algorithm to converge to an adequate policy. We attribute the success of the method to the interplay between experimentation with the world and knowledge-guided refinement. We use this interpretation to make various predictions which are tested and confirmed in further experiments. We note that the results reported here are not specific to this application task; similar results were obtained for a wide range of synthetic target functions.

The remainder of this paper is organized as follows. Section 2 describes our test domain and the specific implementation of the learning method. Section 3 explains the advantages of the phantom induction approach when compared with other methods. Section 4 discusses our initial experiments, including the surprising result that convergence can be non-monotonic if the domain knowledge is inaccurate. Section 5 presents an interpretation which attempts to explain this experimental result. Since the interpretation is based on a number of simplifying assumptions, we conducted further experiments in Sections 6–8 to test the validity of the interpretation. In Section 6 predictions of peak location and sample complexity are made and tested. In Section 7 the algorithm is extended by changing the sampling rate and then by using “windowing” in a fruitless attempt to avoid the non-monotonic convergence—again the results are successfully predicted by the analysis. Section 8 shows that different policy-generation methods yield qualitatively similar results. This verifies another prediction of the interpretation and supports the claim that putting the domain knowledge into the training data allows the choice of inductive algorithm to be made independently of the prior knowledge. Finally, related work is discussed in Section 9.

2. The domain

2.1. Air-hockey

For our experiments we used a simulated air-hockey environment. The task is to guide an incoming puck to a goal position with a single deflection by a circular paddle (see figure 3). In our experiments the goal position, the viscous deflection of the wind, the puck’s point of release, and the puck’s velocity are fixed but the puck’s angle of release varies. The puck impacts the paddle along a horizontal line L across the air-hockey table. The paddle automatically tracks the position of the puck and positions itself to direct the puck back along its incoming path. Just before impact, the paddle is offset horizontally along L so as to deflect the puck in the direction of the goal. The offset is measured negatively to the left and positively to the right. The task is to learn to compute the correct offset as a function of the puck angle.

The puck can miss the goal in various ways. Each of these yields an *observed error*. If the puck is deflected so that it hits the opposite wall, the observed error is defined to be its distance from the goal, measured positively above the goal and negatively below it (see figure 3). If the offset is chosen so badly that the puck does not even reach the opposite wall, the observed error is the distance the puck ended up from the wall added to the distance from the goal to the corner of the opposite wall (see figure 4). This ensures that the error is

Figure 3. Air-hockey.

Figure 4. Observed error is the distance the puck ends up from the goal.

greater than the worst possible error if the puck reaches the opposite wall first. If the offset is chosen even more poorly, so that the paddle misses the puck altogether, the observed error reflects how far the chosen offset was from the feasible range of paddle offsets. If $[-R, R]$ is the range of offsets that result in the paddle hitting the puck, then in case $f(a) > R$ we

add a term $f(a) - R$ to the observed error, and similarly if $f(a) < -R$. Once again this is done so that missing the puck altogether, even by a small amount, results in a larger error than any shot, no matter how bad, that strikes the puck. Note that the chosen offset and the observed error can both be arbitrarily large.

We use a discrete-time simulator of the air-hockey table to compute the “world” puck behavior, taking into account sliding friction, air resistance, and collision dynamics. The correct policy f^* is non-linear; it returns, for each puck angle, the paddle offset needed to deflect the puck to the goal position.

This domain is similar to Moore et al.’s billiards domain (Moore, Atkeson, & Schaal, 1997). It was chosen for its pedagogical appeal; it serves as a compelling and intuitive illustration of the iterated phantom induction approach. We have subsequently applied the phantom induction approach to a more complex bicycle-riding task, which has multi-dimensional state and action spaces and requires making a sequence of decisions to achieve successful control (Brodie & DeJong, 1999). We hope to extend the analysis presented here to that more complicated domain in a future paper.

2.2. The approach

Let f^* be the optimal policy and f_j the system’s current policy. Since f_j imperfectly approximates f^* , errors occur when the system makes decisions using f_j . The system perceives the observed error d in the world—the distance the puck ends up from the goal. Let e^* denote the ideal error function which transforms each observed error d into the correct decision error. This means that using offset $f_j(a) + e^*(d, a)$ instead of offset $f_j(a)$ would have achieved the goal; i.e. $f_j(a) + e^*(d, a) = f^*(a)$. This error function e^* is also highly nonlinear.

Knowledge of e^* would allow points on the target policy f^* to be determined. But prior knowledge of e^* requires a perfect domain theory. A system that possesses an imperfect domain theory will be able to compute only some approximation of e^* , denoted by e . Using this imperfect error function e , the system generates a phantom decision $f_j(a) + e(d, a)$ from each actual decision $f_j(a)$. A new policy f_{j+1} is generated from these phantom decisions, and so on.

If the error approximation e is very inaccurate, the phantom decisions may lie far from the optimal decisions. Thus the central question posed by this approach is: how imperfect can the domain theory be and still guarantee that the approximation e which it generates will be accurate enough to enable the system to converge to a policy f which is the best approximation to f^* ?

2.2.1. Domain knowledge. The prior knowledge used by the system is limited to the fact that the further the paddle offset is from the correct value, the further the puck ends up from the goal. This corresponds to a single qualitatively homogeneous (Forbus, 1984; Kuipers, 1986) explanation. This explanation implies that the offset error is a monotonic function of the observed error. We conducted a series of experiments using the iterated phantom induction algorithm which indicate that this imperfect domain theory is sufficient for robust successful learning.

Our experiments examine error functions of the form $e(d, a) = m \cdot d$, where m is a constant. These functions do not approximate e^* very well. They are the simplest functions that respect the monotonicity constraint imposed by the prior knowledge—they are linear in the first argument (the observed error d) and ignore the second argument (the puck angle a).

The air-hockey domain allows us to explore the implications of a single monotonicity constraint. It is important to understand that this monotonicity constraint is derived analytically from the qualitative explanation. It is not an empirical estimate of the gradient based on performance error. This is a major difference between the knowledge-based approach of phantom induction and feedback error learning (Jordan & Rumelhart, 1992). We discuss this further in the Related Work section.

In a more complex domain different explanations might apply in different regions of the space, generating a variety of constraints. Phantom induction proceeds using the relevant inferred constraint in each region.

2.3. The algorithm

The following algorithm instantiates the general phantom induction approach:

1. Initialize f_0 to zero and the set of training points to the empty set.
2. Set $j = 0$.
3. For $i = 1$ to n :
 - (i) The environment generates a random puck angle a_i .
 - (ii) Apply the current policy to compute the suggested paddle offset; $o_i = f_j(a_i)$.
 - (iii) Observe the puck behavior and note the observed error d_i .
 - (iv) Compute the estimated *decision error* from the observed error using the *error function* e ; this is $e(d_i, a_i)$, the estimated error in the paddle offset.
 - (v) Compute the “phantom” offset $o_i + e(d_i, a_i)$ obtained by adding the estimated offset error to the offset o_i that was tried; this is the phantom decision that the system thinks would have, for puck angle a_i , driven the puck into the goal.
 - (vi) Add the “phantom point” $(a_i, o_i + e(d_i, a_i))$ to the set of training points.
4. Set $j = j + 1$.
5. Generate a new policy f_j from the set of training points.
6. Measure the performance of f_j .
7. If the performance of f_j is satisfactory, exit and return f_j , otherwise return to Step 3.

3. Exploratory experiments

The first set of experiments (Experiments 1–5) explore the algorithm’s sensitivity to the accuracy of the error function e which is used to convert observed errors into estimated

decision errors. The performance of each policy is measured by computing the mean-squared observed error (msE) over an independent, randomly generated, test set of size 100, during which no phantom decisions are made and no learning takes place.

The system collects and remembers all the phantom data points; each iteration adds n phantom points to the training set. In the experiments we employ an instance-based approach (Aha, 1991; Atkeson, 1991). Policy values are computed by convolution of the phantom points, as follows: Given a puck angle a , let the un-normalized weight of each phantom point p_i be $v_i = \exp(-\frac{(a_1 - a)^2}{2\sigma^2})$, where $p_i = (a_i, \tilde{o}_i)$; thus nearby phantom points are weighted more highly, according to a Gaussian centered at a (we used $\sigma = 0.1$). The weights are then normalized to sum to one: $w_i = \frac{v_i}{\sum_{j=1}^{n_j} v_j}$. The paddle offset is computed as a weighted average of all the phantom observations: $f_j(a) = \sum_{i=1}^{n_j} w_i \tilde{o}_i$. In Section 7 we consider other methods of policy generation.

3.1. Experiment 1: Perfect error function

As discussed in Section 2.2, knowledge of the perfect error function e^* would yield the correct decision for each training example. Thus a collection of examples lying on the target policy f^* can be accumulated. Figure 5 shows the results of an experiment which confirm that the system converges rapidly to low-error performance if e^* is known exactly.

3.2. Experiments 2–5: Imperfect error functions

3.2.1. Motivation. Knowledge of the perfect error function e^* is unrealistic. As explained in Section 2.2, the system uses error functions of the form $e(d) = m \cdot d$, which are consistent with its prior domain knowledge. Experiments 2–5 explore the algorithm’s behavior as the slope m of the error approximation function varies. We first consider the best linear approximation to the perfect error function, then a smaller approximation, then a larger approximation, and finally mixtures of something larger and smaller than the best linear approximation. A large linear approximation results in a surprising phenomenon: the algorithm seems initially to diverge, with each policy performing worse than its predecessor. However after some time performance begins to improve until once again low-error performance is achieved. In these and all further experiments n is set to ten. Each policy is sampled for ten inputs, resulting in ten phantom points being added to the training set. A new policy is then generated from the training set.

3.3. Experiment 2: Best linear error function

Figure 6 shows that using the best linear approximation as the error function results in quite rapid convergence, though convergence is of course slower than in the case of perfect error knowledge (figure 5). However, knowledge of the best linear error function is in general also unreasonable, because it depends on the extent and distribution of the still-unseen observation errors. We next investigate the consequences of systematically decreasing the accuracy of the domain knowledge.

Figure 5. Experiment 1: error of successive policies given perfect domain knowledge.

Figure 6. Experiments 2 and 3: error of successive policies for the best linear approximation and a typical underestimating approximation ($m = m^\dagger/5$).

3.4. Experiment 3: Underestimating error function

Suppose the system's linear decision error function underestimates the decision errors (when compared with the best linear approximation). In this case most of the phantom decisions will lie between the actual decision and the correct decision. Figure 6 displays results for a typical underestimate ($m = m^\dagger/5$, where $m^\dagger$ is the slope of the best linear error function). As

expected, performance improves more slowly than before. (The best linear approximation is shown dashed for comparison.)

3.5. Experiment 4: Overestimating error function

Suppose the system's linear decision error function overestimates the decision errors (when compared with the best linear approximation). This is the most serious difficulty for the algorithm, because instability could result if the method “over-compensates” for previous errors. To see this, consider a puck angle a and suppose that the actual offset o_1 tried by the current policy is smaller than the correct offset o^* . The observed error d_1 will correctly indicate that o^* lies above o_1 , but since the estimated decision error is larger than the true decision error, the phantom offset $o_1 + m \cdot d_1$ will be placed above o^* . If an angle near a is now chosen, the actual offset o_2 used by a new policy influenced by the new phantom point lying above o^* may be too large, resulting in a negative observed error d_2 . This correctly indicates that o^* lies below o_2 . However the estimated decision error will again be too large (in magnitude), and the new phantom offset $o_2 + m \cdot d_2$ will lie below o^* . Errors may be magnified by this over-compensation effect and one might expect the algorithm to diverge.

Figure 7 presents the results for an overestimating linear decision error function ($m = 50m^\dagger$). Quite unexpectedly, the algorithm does not diverge. Performance initially worsens, but the msE soon decreases and converges to the same value as in Experiment 2. This behavior is typical for overestimating error functions.

Figure 8 presents results for three successively worse overestimating linear decision error functions ($m = 10m^\dagger$, $m = 50m^\dagger$, $m = 200m^\dagger$) on a logarithmic vertical scale. In

Figure 7. Experiment 4: error of successive policies for an overestimating linear approximation ($m = 50m^\dagger$).

Figure 8. Experiment 4: error of successive policies (on a log scale) for three overestimating linear approximations ($m = 10m^\dagger$, $m = 50m^\dagger$, $m = 200m^\dagger$).

general, we see that if the error function systematically overestimates the true error, the msE initially increases to a peak, but it then decreases until it converges to the same value as in Experiment 2. Overshooting of the phantom points does occur but is then overcome.

This is a rather surprising result which we explain in Section 5. We will also show in Section 6.2 that there is a linear relationship between the accuracy of the approximation and both the location of the peak in the error curve and the number of examples needed for the error to drop below a particular level.

Similar results occur if the actions are not permitted to be arbitrarily large but instead saturate out at some maximum level. However the error curve no longer has a clean peak; instead it smears out somewhat because control is being exercised solely by adjusting the number of phantom points placed at the upper and lower saturation limits.

3.6. Experiment 5: Mixed error function

Imperfect domain knowledge may not produce an error function which systematically over- or under-estimates the decision error. Experiment 5 tests error functions which sometimes underestimate the decision error but at other times overestimate it. There are a large variety of such “mixed” error functions; here we present two typical examples.

Experiment 5a uses a randomly fluctuating error function: $e(d) = m_r \cdot d$ where m_r is uniformly distributed over $(m^\dagger/50, 50m^\dagger)$. Figure 9 presents the results. The observed error does not rise very much when compared with the systematic overestimate in Experiment 4b, and also converges much faster than the systematic underestimate in Experiment 3. Thus using an error function which varies randomly between smaller and larger decision error estimates results in little initial divergence followed by fairly rapid convergence.

Figure 9. Experiment 5a: error of successive policies (on a log scale) for a linear approximation that randomly over or under-estimates the best approximation, compared with typical over and under-estimating linear approximations.

It might be argued that randomly varying error estimates provide an artificially easy obstacle because the over- and underestimates essentially cancel out, leaving a dependable average. To address this concern Experiment 5b uses an error function that systematically overestimates the true decision error in some parts of the space and underestimates it in other parts. Let r denote the proportion of the space in which an overestimate was used ($m = 20m^+$); in the rest of the space an underestimate was used ($m = m^+/20$). We varied r from 0 to 1 in increments of 0.1.

Figure 10 shows the first 3 curves, where overestimation takes place over 0, 0.1 and 0.2 of the space respectively. Curve *over0.0* is of course just a typical underestimation curve. Curves *over0.1* and *over0.2* show that the peak phenomenon begins to occur as soon as some overestimation is introduced.

Figure 11 shows the last 3 curves; curve *over1.0* is the case where overestimation takes place over the entire space. Curves *over0.9* and *over1.0* illustrate that convergence may be faster as the proportion of overestimation increases, because underestimation slows down convergence.

Figure 12 shows the full range of curves for iteration 1–10 and demonstrates that the peak rises systematically as amount of overestimation increases, but the iteration where the peak occurs is unchanged. The following section attempts to explain this result.

4. Interpretation

The results of Experiment 4 may seem somewhat surprising; successful learning occurs even when the decision errors are always overestimated by large amounts, but the error initially rises to a peak before converging. This section advances an interpretation of this

Figure 10. Experiment 5b: error of successive policies for linear approximation functions which overestimate in a small part of the space.

Figure 11. Experiment 5b: error of successive policies for linear approximation functions which overestimate over most of the space.

phenomenon which argues that the expected error behaves approximately as $\frac{m^n}{n!}$, where m is the scale factor (the slope of the error approximation function) and n is the iteration number. For any fixed m , this function initially rises to a peak and then converges to zero as $n \rightarrow \infty$.

Let $e(d) = m \cdot d$ be the linear decision error function, and f^* the ideal function that computes the correct offset. In this section we give an intuitive argument in which we

Figure 12. Experiment 5b: error of successive policies for linear approximation functions which overestimate over increasing parts of the space. The iteration where the error peak occurs is invariant.

assume repeated sampling at the same angle a . In the Appendix we provide a more formal presentation without this assumption. Without loss of generality set $f^*(a) = 1$, which amounts only to a choice of units.

Iteration 1: If the observed error were accurate, $d = f^*(a) - f_0(a)$, which is 1, because the initial policy f_0 is identically zero. Hence the estimated offset error is $e(d) = m \cdot d = m$, and a phantom point is placed at $(a, f_0(a) + e(d)) = (a, m)$.

Iteration 2: There is a phantom point at (a, m) , so $f_1(a) = m$. The offset error is $e(d) = m \cdot (f^*(a) - f_1(a)) = m \cdot (1 - m) \approx -m^2$ if we neglect lower order terms, which is justifiable if m is “large”. A phantom point is placed at $(a, f_1(a) + e(d)) = (a, m - m^2) \approx (a, -m^2)$, neglecting lower order terms.

If m is large the phantom point at $-m^2$ is farther from the correct value (and on the opposite side) than the previous phantom point at m . It appears that the algorithm is diverging.

There are now 2 phantom points at a : (a, m) and $(a, -m^2)$. Assume that the next policy will pass exactly through the average of the phantom points, so $f_2(a) = 1/2 \cdot (m + (-m^2)) \approx -m^2/2$. In the Appendix we will extend this argument to the case where the function value is any weighted average of the phantom points.

Iteration 3: The offset error is now $e(d) = m \cdot (f^*(a) - f_2(a)) \approx m \cdot (1 - (-m^2/2))$. A new phantom point is placed at $(a, f_2(a) + e(d)) \approx (a, m^2/2 + m \cdot (1 + m^2/2)) \approx (a, m^3/2)$, neglecting lower order terms. There are now 3 phantom points at a : (a, m) , $(a, -m^2)$, and $(a, m^3/2)$. The next policy will pass through the average of the phantom points, so $f_3(a) = 1/3 \cdot (m + (-m^2) + m^3/2) \approx m^3/6$.

Repeating this argument, we get:

$$\begin{aligned} f_1(a) &\approx m, \\ f_2(a) &\approx -m^2/2, \end{aligned}$$

$$\begin{aligned} f_3(a) &\approx m^3/6, \\ f_4(a) &\approx -m^4/24, \end{aligned}$$

and in general

$$f_n(a) \approx (-1)^{n+1} \cdot (m^n/n!).$$

So the error, the distance between f_n and f^* , behaves, approximately, on average, as $\frac{m^n}{n!}$. This initially increases to a peak at $n = m$ and then decreases to zero as $n \rightarrow \infty$. This may explain why the error in Experiment 4 first rises and then falls, and why the length and magnitude of the rise increase as the quality of the decision error approximation degrades.

The above argument assumes that policy values are computed by averaging the phantom points, and that successive iterations sample the same angle. In the Appendix these assumptions are relaxed—we assume that a different angle is sampled every iteration and that policy values are computed as a weighted average of the phantom points. We show that the above result holds provided that we replace the behavior of the error by the **expected** behavior of the error; the expected behavior of the error is $\approx \frac{m^n}{n!}$.

This result is valid for any method of policy generation which computes policy values as a weighted average of the phantom observations. If this condition holds then the error should depend only on the accuracy of the domain knowledge, as represented by the value of the scale factor m . The error will be independent of the induction algorithm used to generate the next policy. In the experiments so far convolution was used to compute policy values; Section 7 considers other methods of policy generation.

5. Confirmatory experiments

The above analysis is based on a number of simplifying assumptions, so it is important that its validity be tested. In each of the following experiments we derive a theoretical prediction using the $\frac{m^n}{n!}$ expression for the error. Then empirical results are obtained from the air-hockey domain and compared with the prediction. In Section 5 we will use it to predict the location of the peak and the sample complexity. In Section 6 we will use it to predict the effect of changing the sampling rate and of introducing windowing in an attempt to avoid the initial divergence of the error. In Section 7 we will test whether any inductive algorithm that computes a weighted average of the phantom points can be used to generate the next policy.

5.1. Experiment 6: Peak location

As described above, the most surprising feature of the algorithm's behavior is the fact that the error may rise to extremely large intermediate values before performance converges. Our interpretation was initially motivated as an attempt to understand the fact that the duration

and size of this part of the error curve increases as the accuracy of the domain knowledge degrades, so it is important to verify empirically that it behaves as predicted.

Our interpretation suggests that the error behaves approximately like $m^n/n!$. Thus the iteration where the peak occurs should increase linearly as m increases, since the peak of $m^n/n!$ occurs at $n = m$. We conducted a sequence of experiments in which we measured the location of the peak as a function of the accuracy of the domain knowledge. The accuracy of the approximate domain knowledge is measured by $m/m^\dagger$, where m is the slope of the linear decision error function used and $m^\dagger$ is the slope of the best linear decision error function.

The results (together with a least-squares regression line) are plotted in figure 13. As predicted, the iteration number where the peak occurs is a linear function of the accuracy of the approximation. The $m^n/n!$ approximation will be used to make further testable predictions below.

5.2. Experiment 7: Sample complexity

5.2.1. Motivation. The *sample complexity* is the number of training examples that must be consumed in order for the algorithm to achieve convergence to a particular level of performance. For any learning problem the sample complexity increases with the difficulty of the learning task, but we would like to keep the rate of increase below some reasonable (i.e. polynomial) bound. In our case the sample complexity is a linear multiple of n , the number of iterations, because each iteration consumes a fixed number of training examples. The following analysis suggests that the sample complexity should be a linear function of the accuracy of the error function approximation, as measured by the slope m .

Figure 13. Experiment 6: the iteration where the peak occurs depends linearly on the accuracy of the domain knowledge.

Let $0 < \epsilon < 1$ be arbitrary. Suppose that, for a given value m_0 of the slope, n_0 iterations are required for the error to fall below ϵ ; i.e. $m_0^{n_0}/(n_0)! < \epsilon$. We show that if $m = t \cdot m_0$, then $n = et \cdot n_0$ ensures that $m^n/n! < \epsilon$; i.e. the value of n that makes $m^n/n! < \epsilon$ is a **linear** function of the value of m . We use the fact that $(n/e)^n < n! < n^n$ for all n .

We have:

$$\begin{aligned} m^n/n! &= \frac{(tm_0)^{etn_0}}{(etn_0)!} < \frac{(tm_0)^{etn_0}}{(etn_0/e)^{etn_0}} = \frac{m_0^{etn_0}}{n_0^{etn_0}} < \frac{m_0^{etn_0}}{((n_0)!)^{et}} \\ &= (m_0^{n_0}/(n_0)!)^{et} < \epsilon^{et} < \epsilon. \end{aligned}$$

Hence the number of iterations needed to converge to a particular error should depend linearly on the accuracy of the approximation, measured as before by $m/m^\dagger$.

5.2.2. Results. Figure 14 displays the number of iterations it took the algorithm to converge to an error level of msE = 1 as a function of the accuracy of the approximation.

The number of iterations needed for convergence is seen to depend linearly on the accuracy of the approximation. The same holds for the number of examples because a constant number of examples is being consumed during each iteration. Similar graphs are obtained for any given error level. The empirical results correspond well with the prediction.

Figure 14. Experiment 7: sample complexity depends linearly on the accuracy of the domain knowledge.

6. Modifying the algorithm

6.1. Experiment 8: Windowing

6.1.1. Motivation. We have seen that, in the case of an overestimating error function, the msE curve rises to very large heights before converging to a low value. Our interpretation suggests that this is because the system over-compensates for the mistakes in the initial decisions and these errors accumulate for a while before eventually being overcome. The larger the overestimation, the greater the number of examples required to erase the effect of the early mistakes. It might seem that the rise in the msE could be eliminated or reduced by decreasing the importance of the earlier decisions. If the msE is rising because of the mistaken early decisions, neglecting those points should allow performance to converge to a low msE value without having to go through the lengthy and wasteful process of examining much worse intermediate strategies.

This section examines windowing: instead of using all the phantom points to generate the next policy, only the r most recent phantom points are used. If the earlier points are “bad”, neglecting them should result in improved performance. We will first provide an analysis and then discuss the empirical results.

Suppose the window size is r . Each policy is generated by averaging the r previous phantom points. To simplify the derivation, it will be assumed that each weight actually takes on its expected value.

First suppose that $n \leq r$, where n is the iteration number. Then, following the argument of Section 4, the phantom points lie approximately at

$$m, -m^2, \frac{m^3}{2}, \frac{-m^4}{6}, \dots, (-1)^{n+1} \cdot \frac{m^n}{(n-1)!},$$

yielding a policy value of $f_n(a_n) \approx (-1)^{n+1} \cdot \frac{m^n}{n!}$. In particular this holds for $n = r$.

Now let $n = r + 1$. The policy value of $f_r(a_r) \approx (-1)^{r+1} \cdot \frac{m^r}{r!}$ results in the $(r + 1)$ th phantom point being placed at approximately $(-1)^{r+2} \cdot \frac{m^{r+1}}{r!}$. Then the first phantom point is removed, because the window includes only the r most recent points. There are now r phantom points, located at:

$$-m^2, \frac{m^3}{2}, \frac{-m^4}{6}, \dots, (-1)^{r+1} \cdot \frac{m^r}{(r-1)!}, (-1)^{r+2} \cdot \frac{m^{r+1}}{r!}.$$

The policy value generated by averaging these r points is

$$f_{r+1}(a_{r+1}) \approx (-1)^{r+2} \cdot \frac{m^{r+1}}{r \cdot r!}.$$

Note that this is larger (in magnitude) than the value $(-1)^{r+2} \cdot \frac{m^{r+1}}{(r+1)!}$ obtained in the non-windowing case. This indicates that when the window takes effect the error is larger when compared with the non-windowing case. This is rather surprising.

Now let $n = r + 2$. The policy value of f_{r+1} results in the $(r + 2)$ th phantom point being placed at approximately $(-1)^{r+3} \cdot \frac{m^{r+2}}{r * r!}$. The second phantom point drops out of the window. The r phantom points are now located at:

$$\frac{m^3}{2}, -\frac{m^4}{6}, \dots, (-1)^{r+1} \cdot \frac{m^r}{(r-1)!}, (-1)^{r+2} \cdot \frac{m^{r+1}}{r!}, (-1)^{r+3} \cdot \frac{m^{r+2}}{r * r!}.$$

This yields a policy value of

$$f_{r+2}(a_{r+2}) \approx (-1)^{r+3} \cdot \frac{m^{r+2}}{r^2 * r!}.$$

Continuing in this way, each additional policy introduces an extra factor of m in the numerator (produced by the incorrect decision error function) and an extra factor of r in the denominator (produced by the window). In general

$$\begin{aligned} f_n(a_n) &\approx (-1)^{n+1} \cdot \frac{m^n}{n!} && \text{for } n \leq r \\ f_n(a_n) &\approx (-1)^{n+1} \cdot \frac{m^n}{r^{n-r} * r!} && \text{for } n > r. \end{aligned}$$

Hence the distance between the policy value $f_n(a)$ and the true value $f(a)$ is $\approx (m/r)^n \cdot (r^r/r!)$ for $n > r$. This implies that as $n \rightarrow \infty$ the error should converge to zero if $m < r$ but should diverge if $m > r$; i.e. for the algorithm to converge the window size needs to be made proportionally larger as the amount of over-estimation in the decision error function increases.

In Experiment 8 a fixed scale factor m was tested with various values for the window size r ; it was observed whether or not the algorithm converged. This was repeated for various values of m . The analysis predicts that convergence should occur if $m < r$ and divergence if $m > r$. It also predicts that windowing should not improve performance when compared with the non-windowing case.

6.1.2. Results. Figure 15 presents the results for an overestimating linear decision error function ($m = 20m^\dagger$) with window size 24. After iteration 24 windowing begins to operate as the earliest points pass out of the window. Although these early points are presumably inaccurate (as indicated by the high msE value), ignoring them by windowing results in worse performance. The msE graph displays a second, smaller, hump after iteration 24, rising sharply before beginning to fall again. There are further rises starting at iterations 48, 72 and every multiple of 24 thereafter. (The peak of each secondary rise occurs somewhat after the beginning of the rise.) In general, at multiples of the window size the msE graph begins to rise to another peak and then falls again. In figure 15 the peaks are successively smaller in size, and the msE eventually converges as the peaks damp out.

Our interpretation does not predict this phenomenon of secondary peaks. The data indicate that although the earliest phantom points are very inaccurate, ignoring these points does

Figure 15. Experiment 8a: large window results in convergence.

not lead to better performance. Intuitively, this may be because the system is learning to compensate for these bad points; later phantom points accumulate which, together with the earliest phantom points, yield a particular level of performance. When the earliest phantom points begin to drop out of the window, the remaining phantom points are now over-compensating, generating a less accurate policy. Thus after the window size is reached performance begins to worsen until the system again learns to compensate appropriately. This phenomenon repeats itself at each multiple of the window size, generating successive secondary humps.

Figure 16 presents the results for the same overestimating linear decision error function ($m = 20m^\dagger$) with window size 18, a somewhat smaller window than that used in figure 15. As before, secondary humps occur at multiples of the window size. However in figure 16 the secondary peaks are successively larger in size, so that the msE diverges. In this case the window size is so small that each policy is being generated from an insufficiently small fraction of the total number of available phantom points.

Figure 17 displays the areas of convergence and divergence as a function of the approximation accuracy m and the window size r . In general, if r is somewhat smaller than m the algorithm always diverges. If r is somewhat larger than m the algorithm always converges. For r very close to m , performance is unpredictable: the algorithm might diverge, converge, or oscillate. This corresponds well with the predicted results.

6.2. Experiment 9: Sampling rates

6.2.1. Motivation. The interpretation indicates that if each policy is sampled the same number of times, the “smoothing out” of the inaccurate points results in convergence no matter how badly the linear decision error function overestimates the true decision error.

Figure 16. Experiment 8b: small window results in divergence.

Figure 17. Experiment 8: the window size required for convergence depends linearly on the accuracy of the domain knowledge.

However, if each policy is sampled more often than the previous one, then, as the initial strategies worsen, each generates more phantom decisions than the previous one. If the sampling rate increases fast enough, the number of bad phantom decisions made by the initial strategies should overwhelm the smoothing effect and result in the algorithm diverging.

Let the sampling rate be some function s ; i.e. policy f_n generates $s(n)$ phantom points. The first phantom point is placed at about m and there are $s(1)$ of them; so $f_1(a) \approx \frac{s(1) \cdot m}{s(1)} = m$ as before.

The next phantom point is placed at approximately $-m^2$ and there are $s(2)$ of them, yielding a total of $s(1) + s(2)$ phantom points. The new policy value is

$$f_2(a) \approx \frac{s(1) \cdot m + s(2) \cdot (-m^2)}{s(1) + s(2)} \approx \frac{-s(2) \cdot m^2}{s(1) + s(2)}.$$

The next phantom point is placed at about $\frac{s(2) \cdot m^3}{s(1) + s(2)}$, and there are $s(3)$ of them. There are now $s(1) + s(2) + s(3)$ phantom points in total, so

$$\begin{aligned} f_3(a) &\approx \left(s(3) \cdot \frac{s(2) \cdot m^2}{s(1) + s(2)} \right) / (s(1) + s(2) + s(3)) \\ &= \frac{s(3) \cdot s(2) \cdot m^3}{(s(1) + s(2)) \cdot (s(1) + s(2) + s(3))}. \end{aligned}$$

Hence, in general,

$$f_n(a) \approx (-1)^{n+1} \cdot \frac{(\prod_{i=1}^n s(i)) \cdot m^n}{\prod_{i=1}^n (\sum_{j=1}^i s(j))} \quad (1)$$

Note that if the sampling rate is constant, $s(i) = c$ for all i , then (1) yields

$$f_n(a) \approx (-1)^{n+1} \cdot \frac{c^n \cdot m^n}{\prod_{i=1}^n (i \cdot c)} = (-1)^{n+1} \cdot \frac{c^n \cdot m^n}{c^n \cdot \prod_{i=1}^n i} = (-1)^{n+1} \cdot \frac{m^n}{n!},$$

as before.

6.2.2. Linear sampling. Suppose the sampling rate is linear; i.e. $s(i) = i$. Then (1) yields

$$|f_n(a)| \approx \frac{(\prod_{i=1}^n i) \cdot m^n}{\prod_{i=1}^n (\sum_{j=1}^i j)} = \frac{(\prod_{i=1}^n i) \cdot m^n}{\prod_{i=1}^n (i(i+1)/2)} = \frac{m^n}{\prod_{i=1}^n \frac{i+1}{2}} = \frac{(2 \cdot m)^n}{(n+1)!}$$

Thus in this case we would again expect convergence, although the peaks should be higher and occur about twice as far along as in the constant sampling case.

6.2.3. Quadratic sampling. Suppose the sampling rate is quadratic; i.e. $s(i) = i^2$. Then (1) yields

$$\begin{aligned} |f_n(a)| &\approx \frac{(\prod_{i=1}^n i^2) \cdot m^n}{\prod_{i=1}^n (\sum_{j=1}^i j^2)} = \frac{(\prod_{i=1}^n i^2) \cdot m^n}{\prod_{i=1}^n (i(2i+1)(2i+2)/12)} = \frac{m^n}{\prod_{i=1}^n \frac{(2+1/i)(2i+2)}{12}} \\ &\approx \frac{m^n}{\prod_{i=1}^n \frac{2(2i+1)}{12}} = \frac{m^n}{\prod_{i=1}^n \frac{(i+1)}{3}} = \frac{(3 \cdot m)^n}{(n+1)!} \end{aligned}$$

Figure 18. Experiment 9: If the sampling rate is constant, linearly increasing, or quadratically increasing, the iteration where the peak in the error occurs depends linearly on the accuracy of the domain knowledge.

Again we expect convergence, with even higher peaks occurring about three times as far along as in the constant sampling case.

Results. Figure 18 shows the location of the peak as a function of the accuracy of the approximation for constant, linear, and quadratic sampling rates. The peak's location is accurately predicted as a linear function of the accuracy. The least-squares regression lines that fit the points in the constant, linear, and quadratic case have slopes 0.5, 1 and 1.5 respectively, verifying the prediction that the linear case doubles the peak location while the quadratic case triples it.

6.2.4. Exponential sampling. Suppose next that the sampling rate is exponential; i.e. $s(n) = 2^n$. Then (1) yields

$$\begin{aligned} |f_n(a)| &\approx \frac{(\prod_{i=1}^n 2^i) \cdot m^n}{\prod_{i=1}^n (\sum_{j=1}^i 2^j)} = \frac{(\prod_{i=1}^n 2^i) \cdot m^n}{\prod_{i=1}^n (2^{i+1} - 2)} \\ &= \frac{m^n}{\prod_{i=1}^n (2 - \frac{2}{2^i})} > \frac{m^n}{\prod_{i=1}^n 2} = \left(\frac{m}{2}\right)^n. \end{aligned}$$

Hence we conclude that $f_n(a) \rightarrow \infty$ as $n \rightarrow \infty$ if $m > 2$.

Thus in this case we expect divergence. The number of inaccurate points is increasing so quickly that the smoothing effect of convolution is unable to achieve convergence.

Results. Figure 19 plots the error curve for the case $m = 20m^\dagger$ with an exponential sampling rate (on a logarithmic vertical scale). The error increases at an exponential rate without bound, as predicted.

Figure 19. Experiment 9: an exponentially increasing sampling rate causes the error to increase without bound (log vertical scale).

7. Experiment 10: Policy generation methods

7.1. Motivation

In all the experiments discussed previously policy values were computed by convolution of the phantom training points. We now investigate whether robust convergence depends on this particular method of policy generation. The analysis in Section 4 ascribes the algorithm’s stability to the averaging effect of convolution. Hence any method which also generalizes by “averaging” or smoothing the training points should yield similar results. We tested this by replacing convolution with other generalization techniques and repeating the experiments discussed previously.

The smoothing strategies we have explored are:

- convolution
- regression
- Fourier methods
- neural networks

7.2. Results

Experiments 1–7 were repeated using each of these methods. We obtained similar results in each case. For space reasons we present only the analogs of Experiment 3b: an overestimating linear decision error function ($m = 50m^\dagger$).

7.2.1. Regression. Linear regression is a standard statistical smoothing scheme. We used cubic polynomials; each policy is generated by fitting a cubic to the current set of phantom training points. The fitted cubic specifies the paddle offset for any puck angle. The learning algorithm proceeds exactly as before.

7.2.2. Fourier methods. Fourier methods comprise another family of smoothing techniques. Here we apply a Fourier transform to the phantom training points, filter out the high-frequency components, and use an inverse transform to obtain a smoothed set of points. The policy value is then computed by linear interpolation between adjacent smoothed points.

7.2.3. Neural networks. Neural networks also encode a generalization of their input data. We discretized the continuous input data and, after some experimentation, chose a network structure with 50 input units and one hidden layer of 10 units. We used the back-propagation code from Anguita (1993) to train the network on the set of phantom training points. After back-propagation has run to completion the trained network is used to compute policy values and iterated phantom induction proceeds as before.

The error curves for all four smoothing techniques (convolution, regression, Fourier methods, and neural networks) are shown together (on a logarithmic scale) in figure 20. Their similar shapes are clearly apparent. All the curves rise to a peak at about the same iteration, and then converge. All the phenomena predicted by the analysis, including peak location, sample complexity, sampling rate, and windowing, were experimentally verified for all the inductive techniques. This confirms that a general inductive approach is all that is required and that the error depends primarily on the accuracy of the domain knowledge and not on the choice of inductive algorithm used to generate the next policy.

Figure 20. Experiment 10: different inductive algorithms produce qualitatively similar results (The error curves shown here are for the case $m = 50m^{\dagger}$.) Performance depends primarily on the accuracy of the domain knowledge, not on the inductive algorithm used.

8. Related work

Reinforcement learning (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998) offers many approaches to learning the optimal policy, both model-based and model-free. However it can be difficult for these methods to effectively utilize domain knowledge; the human expert must usually express his/her knowledge in the form of a learning bias internal to the system. In a direct comparison on the same bicycle-riding task, iterated phantom induction improved significantly on the performance of the reinforcement learning algorithm of Randlov and Alstrom by effectively exploiting domain knowledge (Brodie & DeJong, 1999).

Neural nets can be viewed as forming approximations to numeric functions. Two notable approaches that support the incorporation of explicit background knowledge are KBANN (Shavlik & Towell, 1990) and EBNN (Thrun & Mitchell, 1993). The distinguishing feature of iterated phantom induction is that the knowledge is used to adjust the observed data, so that training is based not on the actual examples but on phantom examples. The domain knowledge does not directly affect the hypothesis-generation process.

The ALVINN system (Pomerleau, 1993) also creates synthetic training examples from actual training examples using domain knowledge. However their approach is rather different from phantom induction. ALVINN obtains actual training examples, including the correct decision, from humans. However these data points are unrepresentative of the complete task, because they do not include enough examples of situations which humans do not get into very often. So ALVINN generates an additional synthetic training examples from each actual training example. The world state of each actual training example is altered, and the correct decision for the new state is computed from the human decision for the old state by using a mathematical model of how this decision should be changed as a result of the change in state.

A similar approach has been adopted to using prior knowledge to expand the training set by creating *virtual examples* (Niyogi, Girosi, & Poggio, 1998). The prior knowledge takes the form of knowing that the target function is invariant under certain transformations; e.g. rotational symmetry. If $(x, f(x))$ is an actual example lying on the unknown target function f , then $(Tx, y_T(f(x)))$ is also an example from f , where T is a transformation. Thus, given actual positive examples, additional examples can be created which are also positive examples.

These methods transform actually experienced successful decisions using strong prior knowledge to obtain successful decisions for other states that have not actually been experienced. These synthetic examples are then added to the training set, thereby obtaining a more representative collection of training data. In contrast, phantom induction transforms actually experienced unsuccessful decisions using prior knowledge in the form of a weak qualitative domain theory to obtain phantom decisions for the same states that have already been experienced. These decisions will probably also be unsuccessful because the domain theory is imperfect, but training proceeds by re-labeling them as actually experienced successful decisions.

Finally, “forward” and “inverse” models offer alternative approaches to learning from error feedback (Jordan & Rumelhart, 1992). If S is the set of world states and A the set of actions, let $d(s, a)$ be the error resulting from taking action a in state s . The forward model

approach learns the error as a function of all the state/action pairs. A surface $e(s, a)$ is fit to the (s, a, d) data points. When actions are required they are chosen to achieve zero error; given s , root-finding is used to select a such that $e(s, a) = 0$. The inverse model approach learns the action as a function of the state and the desired error level; a surface is fit to the (s, d, a) data points, and when actions are required d is set to 0 to select a . By contrast iterated phantom induction learns from training examples of the form $(s, \tilde{a})$, where $\tilde{a}$ is the phantom action $\tilde{a} = a + d(s, a)$.

The forward and inverse model approaches introduce an additional dimension which requires learning those parts of the surface where $d \neq 0$. Yet we are only interested in that part of the surface where $d = 0$. The system must learn to predict all the different ways the task may fail to be performed correctly, and then selects actions so that the failures do not occur. This additional commitment may in fact result in convergence to a non-optimal policy, because the system's flexibility may be exhausted fitting the parts of the surface that are not actually needed. We show below that this can occur even if the observed error coincides with the true error; i.e $d(s, a) = f^*(s) - a$.

Let E be the collection of surfaces $e(s, a)$ that are used to fit the data. For notational simplicity the forward model is used in the rest of the discussion but a similar argument applies to any higher-dimensional approach. For any $e \in E$ define f_e to be the resulting policy; i.e. $f_e(s) = a$ if and only if $e(s, a) = 0$. Assume that this is uniquely defined; if there are multiple roots this may constitute an additional problem for the forward model approach, but that issue is ignored here. Recall that f^* denotes the optimal policy.

Suppose that the best approximation from E to the true error surface is some $e^*(s, a)$, which yields policy f_{e^*} . Let f_* be the best approximation to the optimal policy achievable by E —i.e. of all $f_e, e \in E$, f_* is the closest to the optimal policy f^* . Then it will not generally be the case that $f_{e^*} = f_*$. For although $|e^*(s, a) - d(s, a)| < |e(s, a) - d(s, a)| \forall e \in E$, where the norm is computed over the entire surface, nevertheless there may be some $e \in E$ such that $|f_{e^*} - f^*| > |f_e - f^*|$, where the norm is computed over all the states in S . Thus, even if the method converges to e^* , the best approximation to the entire error surface, the policy may fail to converge to f_* , the best achievable solution.

In the forward model case this occurs because the roots of e^* may not be the best approximation from E to the roots of d . More generally, any higher-dimensional approach may fail to converge to the best solution because it may exhaust its flexibility fitting parts of the surface where $d \neq 0$, thus preventing it from fitting the surface near $d = 0$ as well as it could.

The advantage of the iterated phantom induction approach is that it focuses the training examples on the region of interest. It is not necessary to learn all the different ways of failing to perform the task correctly. Iterated phantom induction exploits the training examples to achieve the overall learning goal directly; it learns only that part of the surface where $d = 0$ —how to perform the task successfully.

9. Conclusions

The iterated phantom induction approach constructs “phantom” training data through a knowledge-based alteration of the actual world observations. Each observed example is

systematically perturbed by an amount and direction dictated by the explanation of its failure. Induction is performed to generate the candidate policy which best describes the phantom training data. The selected policy is then employed to solve additional problems, resulting in more phantom data which augment the training set, and the process repeats.

It has been claimed that the disadvantage of learning the policy alone is that “a strategy for updating the policy requires extra fore-knowledge of the structure of the world: if we apply an action which gives sub-standard behavior, we need to know which direction to alter the mapping so that next time the adjusted action will produce improved behavior” (Moore, Atkeson, & Schaal, 1995). Iterated phantom induction is premised precisely on the idea that such “extra fore-knowledge” is often available and should be exploited wherever possible.

We believe that the learning process should use as much prior knowledge as possible. The difficulty is how to gracefully integrate the prior knowledge with what is learned from empirical observations, particularly when the knowledge is approximate and imperfect. Phantom induction achieves this by using a qualitative form of domain knowledge. This form of knowledge is not only appropriate for generating phantom decisions; in addition human experts can usually express their domain knowledge easily and naturally in qualitative terms. With phantom induction the human knowledge does not have to be transformed into the bias language of a particular learning algorithm in order to be effectively utilized by the machine learning system.

The main result of this paper is that iterated phantom induction converges quickly to a good decision policy. Domain knowledge is required for the approach, but a simple qualitative domain theory suffices for convergence. A slope must be chosen to quantify the qualitative error function. Correctly approximating the slope results in quickest convergence. A slope which is too small attenuates the error feedback and slows learning. A slope which is too large magnifies the error, initially causing poor policies to be used but, even when the observed error grows extremely large, further iterations overcome the error growth and the algorithm converges to an accurate decision policy.

While our initial results are promising, much remains to be done. Future work will consider more complex domains in which the explanations play a more prominent role. It is noteworthy that the background knowledge used by phantom induction is employed in perturbing the training data and does not permeate the induction algorithm itself. This suggests a way of incorporating prior knowledge into a broad class of conventional induction algorithms.

10. Appendix

Here we present a more formal version of the argument in Section 4 that the error behaves approximately as $m^n/n!$, where n is the iteration count and m the amount of overestimation in the error approximation.

Let $e(d) = m \cdot d$ be the linear decision error function, f^* the optimal function that computes the correct offset, and $a_0, a_1, \dots$, the successive puck angles generated by the world. Without loss of generality set $f^*(a_0) = 1$, which amounts only to selecting the units in which the observed error is measured. (If this assumption is not made then m should be replaced in what follows by $\frac{m}{m^*}$.)

Recall that the initial policy f_0 is identically 0. Note also that if m is sufficiently large and f^* is bounded on the state space A , then we can regard all terms of the form $f^*(a)$, $a \in A$, as negligible when compared with powers of m . In particular, this holds if f^* is continuous and A is a closed bounded interval, as in the air-hockey domain.

In general each phantom point is given by

$$\begin{aligned} p_j &= f_{j-1}(a_{j-1}) + m \cdot d_{j-1} \\ &= f_{j-1}(a_{j-1}) + m \cdot (f^*(a_{j-1}) - f_{j-1}(a_{j-1})), \end{aligned}$$

where d_j is the true error at iteration j . Suppose that policy values are computed as a weighted average of the phantom observations:

$$f_j(a_j) = \sum_{i=1}^j w_{j,i} \cdot p_i, \text{ where } \sum_{i=1}^j w_{j,i} = 1 \text{ for any iteration } j.$$

So: $f_n(a_n) - f^*(a_n)$

$$\begin{aligned} &= \sum_{j_1=1}^n w_{n,j_1} \cdot p_{j_1} - f^*(a_n) \\ &= \left[\sum_{j_1=1}^n w_{n,j_1} \cdot (f_{j_1-1}(a_{j_1-1}) + m \cdot d_{j_1-1}) \right] - f^*(a_n) \\ &= \left[\sum_{j_1=1}^n w_{n,j_1} \cdot \left(f_{j_1-1}(a_{j_1-1}) + m \cdot \left(f^*(a_{j_1-1}) - \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot p_{j_2} \right) \right) \right] - f^*(a_n) \\ &= \left[m \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(f^*(a_{j_1-1}) - \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot p_{j_2} \right) \right] + O(1) \\ &= \left[m \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(f^*(a_{j_1-1}) - \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot (f_{j_2-1}(a_{j_2-1}) + m \cdot d_{j_2-1}) \right) \right] + O(1) \\ &= \left[m^2 \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(- \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot (f^*(a_{j_2-1}) - f_{j_2-1}(a_{j_2-1})) \right) \right] + O(m) \\ &= \left[m^2 \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(- \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot \left(f^*(a_{j_2-1}) - \sum_{j_3=1}^{j_2-1} w_{j_2-1,j_3} \cdot p_{j_3} \right) \right) \right] + O(m) \\ &= \left[m^2 \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(- \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot \left(f^*(a_{j_2-1}) \right. \right. \right. \\ &\quad \left. \left. \left. - \sum_{j_3=1}^{j_2-1} w_{j_2-1,j_3} \cdot (f_{j_3-1}(a_{j_3-1}) + m \cdot d_{j_3-1}) \right) \right) \right] + O(m) \end{aligned}$$

$$\begin{aligned}
&= \left[m^3 \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(- \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \right. \right. \\
&\quad \left. \left. \cdot \left(- \sum_{j_3=1}^{j_2-1} w_{j_2-1,j_3} \cdot (f^*(a_{j_3-1}) - f_{j_3-1}(a_{j_3-1})) \right) \right) \right] + O(m^2) \\
&= \dots \left[m^n \cdot \sum_{j_1=1}^n w_{n,j_1} \cdot \left(- \sum_{j_2=1}^{j_1-1} w_{j_1-1,j_2} \cdot \left(- \sum_{j_3=1}^{j_2-1} w_{j_2-1,j_3} \cdot \dots \right. \right. \right. \\
&\quad \left. \left. \cdot \left(- \sum_{j_n=1}^{j_{n-1}-1} w_{j_{n-1}-1,j_n} \cdot d_{j_{n-1}} \right) \cdot \dots \right) \right] + O(m^{n-1}) \\
&= [m^n \cdot w_{n,n} \cdot (-w_{n-1,n-1}) \cdot (-w_{n-2,n-2}) \cdot \dots \cdot (-w_{2,2}) \cdot (-w_{1,1}) \cdot d_0] + O(m^{n-1}) \\
&= (-1)^{n+1} \cdot [m^n \cdot w_{n,n} \cdot w_{n-1,n-1} \cdot w_{n-2,n-2} \cdot \dots \cdot w_{2,2} \cdot w_{1,1}] + O(m^{n-1})
\end{aligned}$$

Hence $d_n = f_n(a_n) - f^*(a_n) \approx (-1)^{n+1} \cdot \prod_{i=1}^n w_{i,i} \cdot m^n$.

The behavior of the weights w_{ii} , $i = 2, \dots, n$, must now be investigated. At iteration n , each weight w_{ni} , $i = 1, \dots, n$, determines the contribution of the i 'th phantom point p_i to the policy value $f_n(a_n)$. In general, the value of the weight w_{ni} depends not only on the n 'th sample point a_n and the n phantom points p_i (and thus on $a_0, a_1, \dots, a_{n-1}$), it also depends on the algorithm that is used to generate the next policy from the phantom training points.

In the initial experiments convolution is used. Each p_i is given an unnormalized weight computed by centering a Gaussian at a_n and using the horizontal distance of p_i from a_n as the index into the Gaussian—thus phantom points closer to a_n are weighted more heavily. The weights are then normalized to sum to one, so that each policy value is an appropriately weighted sum of the phantom decisions.

Since the phantom points are located at the previous sample points a_i , $i = 0, \dots, n-1$, which are generated by the world according to some fixed but unknown distribution, the w_{ni} are identically distributed random variables. Applying the expected value $E(\cdot)$ to $\sum_{i=1}^n w_{ni} = 1$ yields

$$\sum_{i=1}^n E[w_{ni}] = n \cdot E[w_{ni}] = 1.$$

Hence

$$E[w_{ni}] = 1/n, \quad i = 1, \dots, n.$$

This simply asserts that if n identically distributed weights sum to 1 then their average value is $1/n$.

Since the weights on successive iterations are independent of each other,

$$E \left[\prod_{i=2}^n w_{ii} \right] = \prod_{i=2}^n E[w_{ii}] = \prod_{i=2}^n \frac{1}{i} = \frac{1}{n!}.$$

Thus we conclude that

$$E[d_n] \approx (-1)^{n+1} \cdot \frac{m^n}{n!}.$$

Acknowledgment

This research was sponsored by the Office of Naval Research under grant N00014-94-1-0684.

References

- Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1), 37–66.
- Anguita, D. (1993). Neural network backpropagation. Software available from risc6000.dibe.unige.it, University of Italy.
- Atkeson, C. G. (1991). Using locally weighted regression for robot learning. *IEEE International Conference on Robotics and Automation*, pp. 958–963.
- Brodie, M. & DeJong, G. (1998). Iterated phantom induction: A little knowledge can go a long way. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 665–670.
- Brodie, M. & DeJong, G. (1999). Learning to ride a bicycle using iterated phantom induction. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 57–66.
- DeJong, G. (1994). Learning to plan in continuous domains. *Artificial Intelligence*, 65, 71–142.
- Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24, 85–168.
- Genesereth, M. & Nilsson, N. (1987). *Logical foundation of artificial intelligence*. Los Altos, CA: Morgan Kaufman.
- Jordan, M. & Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16, 307–354.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 237–258.
- Kuipers, B. J. (1986). Qualitative simulation. *Artificial Intelligence*, 29, 289–338.
- McCarthy, J. & Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence*, (Vol. 4, pp. 463–502). Edinburgh: University of Edinburgh Press.
- Moore, A. W., Atkeson, C. E., & Schaal, S. A. (1995). Memory based learning for control. *CMU-RI-TR-95-18*.
- Moore, A. W., Atkeson, C. E., & Schaal, S. A. (1997). Locally weighted learning for control. *AI Review*, 11.
- Niyogi, P., Girosi, F., & Poggio, T. (1998). Incorporating prior information in machine learning by creating virtual examples. *IEEE Proceedings on Intelligent Signal Processing*, 86, 2196–2209.
- Pomerleau, D. A. (1993). Knowledge-based training of artificial neural networks for autonomous robot driving. In J. Connell & S. Mahadevan (Eds.), *Robot Learning*. Boston, MA: Kluwer Academic Publishers.
- Rich, E. & Knight, K. (1991). *Artificial intelligence*. New York: McGraw-Hill.
- Russell, S. & Norvig, P. (1995) *Artificial intelligence: A modern approach*. New Jersey: Prentice-Hall.
- Shavlik, J. & Towell, G. (1990). An approach to combining explanation-based and neural learning algorithms. In J. Shavlik & T. Ditterich (Eds.), *Readings in Machine Learning*. San Mateo: Morgan Kaufman.
- Sutton, R. & Barto, A. (1988). *Reinforcement learning: An introduction*. MA: MIT Press.
- Thrun, S. & Mitchell, T. (1993). Integrating inductive neural network-learning and explanation-based learning. In *Thirteenth International Joint Conference on Artificial Intelligence*.

Received June 19, 1998

Revised February 26, 2001

Accepted March 1, 2001

Final manuscript march 19, 2001