Two Kinds of Training Information for Evaluation Function Learning[1]

PAUL E. UTGOFF (UTGOFF@CS.UMASS.EDU)
JEFFERY A. CLOUSE (CLOUSE@CS.UMASS.EDU)
Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003 U.S.A.

Abstract

This paper identifies two fundamentally different kinds of training information for learning search control in terms of an evaluation function. Each kind of training information suggests its own set of methods for learning an evaluation function. The paper shows that one can integrate the methods and learn simultaneously from both kinds of information.

1 Introduction

This paper focuses on the problem of learning search control knowledge in terms of an evaluation function. The conclusion is that one can and should seek to learn from all kinds of training information, rather than be concerned with which kind is better than another. Many kinds of information are often available, and there is no point in ignoring any of them.

An evaluation function provides a simple mechanism for selecting a node for expansion during search. An evaluation function maps each state to a number, thereby defining a surface over the state space that can be used to guide search. If the number represents a reward, then one can search for a sequence of control decisions that will lead to the highest foreseeable payoff. Similarly, if the number represents a cost or penalty, then one searches for a minimizing sequence.

2 Sources of Training Information

There are currently two known fundamental sources of training information for learning an evaluation function. The first is the future payoff that would be achieved by executing a sequence of control decisions from a particular starting point (Samuel, 1963; Lee & Mahajan, 1988). Sutton (1988) has illustrated via his temporal difference (TD) methods that one can learn to predict the future value for a state by repeatedly correcting an evaluation function to reduce the error between the local evaluation of a state and the backed-up value that is determined by forward search. This is similar to an idea of Samuel (1963), but Sutton has broadened it considerably and related it to several other lines of thought.

The second source of training information is identification of the control decision made by an expert, given a particular state. In the literature, such an instance of an expert choice is typically called a *book move* (Samuel, 1967), but it need not have been recorded in a book. Instead, one can simply watch an expert in action, or ask an expert what to do in a particular situation, and thereby obtain the control decision that the expert would make. Whenever an expert's choice is available, one would like to be able to learn from it. Such a choice is the result of the expert's prior learning, and therefore should be quite informative. Indeed, learning to make the same choices as an expert is a sensible approach to building an expert system.

3 State Preference Methods

When making a control decision based on the value of each successor state, the exact value of a state is irrelevant with respect to making the choice. Only the relationship of two values is needed for the purpose of identifying the one with the higher value. The objective is to identify the most preferred state and then move to it. Given that a control decision does not depend on the particular values returned by an evaluation function, one does not need to learn an exact value for each state. One needs only to learn a function in which the relative values for the states are correct.

Whenever one infers, or is informed correctly, that state a is preferrable to state b, one has obtained information regarding the slope for part of a correct evaluation function. Any surface that has the correct sign for the slope between every pair of points is a perfect evaluation function. An infinite number of such evaluation functions exist, under the ordinary assumption that state preference is transitive. One would expect the task of finding any one of these evaluation

[1] In *Proceedings of the Ninth International Conference on Artificial Intelligence*, July 1991, pp 596-600.

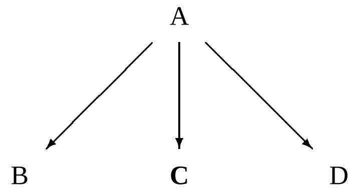

Figure 1. One Ply of Search.

functions to be easier than the task of finding a particular evaluation function.

Because one wants to learn to select a most preferred state from a set of possible successors, one should be able to learn from examples of such choices (Utgoff & Saxena, 1987; Utgoff & Heitman, 1988). By stating the problem of selecting a preferred state formally, and expanding the definitions, a procedure emerges for converting examples of state preference to constraints on an evaluation function. One can then search for an evaluation function that satisfies the constraints, using standard methods. We refer to a method that learns from such constraints as a *state preference (SP)* method.

Assume that a state x is described by a conjunction of d numerical features, represented as a d-dimensional vector $\mathbf{F}(x)$. Also assume that the evaluation function $H(x)$ is represented as a linear combination $\mathbf{W}^T\mathbf{F}(x)$, where $\mathbf{W}$ is a column vector of weights, and $\mathbf{W}^T$ is the transpose of $\mathbf{W}$. Then one would compare the value of a state C to a state B by evaluating the expression $H(C) > H(B)$. In general, one can define a predicate $P(x,y)$ that is true if and only if $H(x) > H(y)$, similar to Huberman's (1968) hand-crafted *better* and *worse* predicates. One can convert each instance of state preference to a constraint on the evaluation function by expanding its definitions. For example, as shown in Figure 1, if state C is identified as best, one would infer constraints $P(C,B)$ and $P(C,D)$. Expanding $P(C,B)$, for example, leads to:

$$P(C,B)$$

$$H(C) > H(B)$$

$$\mathbf{W}^T\mathbf{F}(C) > \mathbf{W}^T\mathbf{F}(B)$$

$$\mathbf{W}^T(\mathbf{F}(C) - \mathbf{F}(B)) > 0$$

The difference between the two feature vectors is known, leaving $\mathbf{W}$ as the only unknown.

By expanding all instances of state preference in the above manner, one obtains a system of linear inequalities, which is a standard form of learning task for a variety of pattern recognition methods (Duda & Hart, 1973), including perceptron learning and other more recent connectionist learning methods. Note that these instances of state preference are expressed as d-dimensional vectors, meaning that learning from pairs of states is no more complex than learning from single states. This is in contrast to Tesauro (1989), where both states are given as input to a network learner.

It is worth noting that Samuel's (1963, 1967) method for learning from book moves is an SP method. When learning from book moves, Samuel computed a correlation coefficient as a function of the number of times L (H) that the feature value in a nonpreferred move was lower (higher) than the feature value of the preferred move. The correlation coefficient for each feature was $\frac{L-H}{L+H}$, and was used directly as the weight in his evaluation function. The divisor $L+H$ is constant for all features, serving only as a normalizing scalar. Thus the total $L-H$ is a crude measure of how important the feature is in identifying a state preferred by an expert.

4 Illustration

This section illustrates a TD method and an SP method, applied individually to the same problem. The purpose is to ground the discussion of the previous sections, and to provide some indication of the relative strength of the two kinds of training information. One already expects that state preference information is stronger than temporal difference information, so the point of interest is really how much stronger. The comparison is not a contest because there is no need pick a winner. Each method learns from a different kind of training information, which raises the issue of how to learn from both sources. One can and should strive to learn from all the training information, regardless of its source.

Table 1. T1 as Best-First Search

```
1. open←(start), closed←nil
2. s←cheapest(open)
3. if solution(s), stop
4. put s onto closed, c←successors(s)
5. if training, do TD adjustment
6. open←append(c − closed,open)
7. re-evaluate all nodes on open
8. goto 2
```

For the TD method and the SP method described below, it is important to note that we have instantiated TD and SP in a particular way, and have coupled each one with the best-first search algorithm. TD and SP methods are generic, and are independent of any particular search algorithm that makes use of the learned evaluation function. To keep this distinction in mind, we refer to the two example progams below as T1 and S1.

4.1 The Task Domain

Although we are experimenting with TD and SP methods in larger domains, as mentioned in the final section, we have selected the smaller Towers-of-Hanoi domain for pedagogical purposes. The main reason for this choice is that the domain is characterized by a small state space, which can be controlled by varying the number of disks. The small domain makes it possible to implement a simple expert, which serves as a source of state preference information for the S1 program.

The semantics of this problem makes it more natural to think of the value of a state as a measure of remaining cost. Accordingly, the goal state has a cost of 0. The problem-solving program will be looking for a state with a low cost, and one state will be preferred to another if its cost is lower. Thus, for any two instances x and y, expanding $P(x,y) \leftrightarrow H(x) < H(y)$ leads to a constraint of the form

$$\mathbf{W}^T(\mathbf{F}(x) - \mathbf{F}(y)) < 0.$$

4.2 A Temporal Difference Method

The T1 program learns from temporal differences, as part of the best-first search algorithm shown in Table 1. The value backed up from the children of the node just expanded is the value of the lowest cost child plus δ, $\delta = 1$. This backed-up value is the desired value of the parent with respect to the children, and the learning mechanism adjusts the weights **W** so that the evaluation for the parent state is closer to this backed-up value. Because the value of the goal state is defined to be 0, the evaluation function is being trained to predict the distance remaining from a state to the goal.

The error correction rule is a form of the well-known absolute error correction rule (Nilsson, 1965; Duda & Hart, 1973), which calculates the amount of correction needed to remove the error. One solves

$$(\mathbf{W} + c\mathbf{F}(x))^T\mathbf{F}(x) = \textit{backed-up value}$$

for c and then adjusts **W** by

$$\mathbf{W} \leftarrow \mathbf{W} + \eta c\mathbf{F}(x)$$

so that $\mathbf{W}^T\mathbf{F}(x)$ is closer to the intended value. The learning rate η is 0.1. Over time, a series of such corrections to **W** should result in an evaluation function that is predictive of minimum cost to the goal state, assuming such a fit can be approximated well in the given feature space.

4.3 A State Preference Method

The S1 program learns from state preferences, as part of the best-first search algorithm shown in Table 2. For training, the expert's choice is simulated by brute force search for the optimal move. From the expert's choice, the

Table 2. S1 as Best-First Search

```
1. open←(start), closed←nil
2. if training, [s←expertbest(open), do SP adjustments]
   else s←cheapest(open)
3. if solution(s), stop
4. put s onto closed, c←successors(s)
5. open←append(c − closed,open)
6. re-evaluate all nodes on open
7. goto 2
```

Table 3. Results for T1 and S1

Method		3 dsks	4 dsks	5 dsks
T1	adjustments	1,308	10,301	—
	trials	107	496	—
	queries	0	0	0
	halt	opt	opt	cyc
	expansions	8	16	56
S1	adjustments	5	9	11
	trials	1	2	1
	queries	7	30	31
	halt	opt	opt	opt
	expansions	8	16	32

algorithm infers that the selected state is to be preferred to each of the nonselected states. From each such pair of states, S1 infers a constraint on the weight vector **W** expressed as a linear inequality. If the constraint is not satisfied, then the weight vector **W** is adjusted.

As with T1, the correction rule is a form of the absolute error correction rule. One solves

$$(\mathbf{W}+c(\mathbf{F}(x)-\mathbf{F}(y)))^T(\mathbf{F}(x)-\mathbf{F}(y)) = -1$$

for c and then adjusts **W** by

$$\mathbf{W} \leftarrow \mathbf{W}+c(\mathbf{F}(x)-\mathbf{F}(y)).$$

Adjusting the weights so that the weighted difference is -1 corresponds to wanting the selected state to evaluate to at least one less than a nonselected state, but any negative value will suffice in order to become correct for the inequality.

4.4 Discussion

Each program was trained repeatedly until either it was able to solve the Towers of Hanoi problem optimally or it had a sequence of weight vectors that was cycling. For each program, the cost of training was measured three ways: by the total number of adjustments to the weight vector **W**, by the number of times the program was trained on the problem (a trial), and by the number of times the expert was queried for its control decision. Table 3 shows S1 requires fewer weight adjustments and fewer trials than T1, but at the expense of querying the expert. For the 5-disk problem, S1 learned to solve the problem optimally, but T1 was unable to do so. “Expansions” is the number of node expansions that occurred when the program solved the problem after it had completed its training.

The problem faced by T1 is to learn an exact value for each state, which is an impossible task in this case because the desired values are not co-planar in the given feature space. It is for this reason that one needs best-first search instead of simple hill-climbing. S1 needs only to learn a value for each state that causes the relationships of the values of the states to be correct. This too is an impossible task in the given feature space, but it appears easier for a learning algorithm to try to satisfy the less demanding constraints of relative values than exact values.

The features for describing a state are a function of the number of disks. For example, Table 4 shows the ten Boolean features for the 3-disk problem. Disk 3 is the largest, and Peg 3 is the goal peg. In general, for the *n*-disk problem, there are $O(3^n)$ states and, in the representation for T1 and S1, $O(n^2)$ features.

Table 4. Features for the 3-Disk Problem.

Feature
Is Disk 3 on Peg 3?
Is Disk 2 at its desired location?
Is Disk 1 at its desired location?
Is Disk 2 on Disk 3?
Is Disk 1 on Disk 3?
Is Disk 1 on Disk 2?
Is Disk 2 on Peg 3?
Is Disk 1 on Peg 3?
Is Disk 3 clear?
Is Peg 3 empty?
Threshold Constant 1

Table 5. I1 as Best-First Search

```
1. open←(start), closed←nil
2. lasterror← 0.0
3. if training and lasterror $> \beta$,
               [s←expertbest(open), do SP adjustments]
   else s←cheapest(open)
4. if solution(s), stop
5. put s onto closed, c←successors(s)
6. if training, lasterror← $|$tderror$|$, do TD adjustment
7. open←append(c $-$ closed,open)
8. re-evaluate all nodes on open
9. goto 3
```

5 Integrating TD and SP Methods

This section discusses the relationship between TD and SP methods, and shows that both kinds of methods can work together in learning one evaluation function.

5.1 Relationship of TD and SP Methods

TD methods learn to predict future values, whereas SP methods learn to identify preferred states. For TD methods, training information is propagating vertically up the search tree. For SP methods, the training information is propagating horizontally among siblings.

Semantically, the two kinds of methods are compatible because the evaluation function is of the same form, and serves the same purpose of allowing identification of a best state. One can adjust the weights **W** so that the value of a state is predictive of its eventual payoff, and one can also adjust **W** so that the relative values among the states become correct. Thus, in terms of the semantics of the learning, one can simply apply both kinds of error correction to the same evaluation function simultaneously without fear that they are incongruous. However, a practical problem that can arise is that the expert might be fallible, putting the two sources of training information in conflict to some degree. This issue is discussed below.

5.2 An Integrated Method

In the same way that TD and SP are each a class of methods, there are many combinations of methods that would produce an integrated TDSP method. We present one such method here, instantiated in a program that we refer to as I1. As noted above, it is permissible to apply a TD method and an SP method to the same evaluation function. Thus, the I1 program, shown in Table 5, is the union of the T1 and S1 programs, with the addition of a dynamic test for when to ask the expert for its control decision.

Table 6. Results for I1

Method		3 dsks	4 dsks	5 dsks
I1	adjustments	35	131	409
	trials	1	1	1
	queries	6	14	24
	halt	opt	opt	opt
	expansions	8	16	32

A TD method can be employed very easily in an unsupervised manner whenever a node is expanded. An SP method relies on an expert, which can be a human or a search procedure. At issue is when to obtain state preference information from the expert. If one can simply observe the expert passively, then there is no apparent expense in obtaining such information. For I1 however, we assume that one must query the expert to obtain state preference information, and that one would like make such queries as seldom as possible. As an extreme, one could avoid querying the expert altogether, and learn only from the TD information. However, expert preferences provide strong training information and should be considered when available.

The I1 program queries the expert whenever the magnitude of the previous TD error is above β, with $\beta = 0.9$. The effect is that the expert exerts great influence early in the training, but is progressively ignored as the evaluation function becomes more accurate.

Table 6 shows the same measures for I1 as those given for T1 and S1. I1 learned to solve all three versions of the problem optimally, with fewer weight adjustments than T1, and fewer queries to the expert than S1. For the 4-disk problem, I1 learned the task in one trial, which is fewer than for either S1 or T1.

The I1 program increasingly ignores the expert as the evaluation function is learned. This is a desirable characteristic in terms of gaining autonomy, but it is also desirable if the expert is imperfect, e.g. human. One can learn rapidly from the expert, and then let TD training correct any flaws that may have crept in from believing the expert. However, it may happen that TD error might temporarily increase without expert input, causing the expert to be drawn back into the training, thereby preventing the improvement that might occur otherwise. The I1 program illustrates just one scheme for integrating TD and SP methods. We are continuing to examine the issue of how to integrate these sources of training information profitably.

6 Conclusion

We have identified two different kinds of training information for learning evaluation functions, and described their relationship. For state preference, we have shown that one can convert instances of state preference to constraints on an evaluation function, and that one can learn an evaluation function from such information alone. We have taken the view that one should be able to learn from all sources of training information, and not be diverted by arguments that one is to be favored over another. We have observed that it is semantically correct to apply a TD method and an SP method simultaneously to the learning of one evaluation function. Finally, we presented a specific method that integrates both approaches, and demonstrated that the two can indeed work together profitably.

Although we have chosen a simple problem for illustration, the issues that motivated this work arose while studying the effectiveness of TD and SP methods in the game of Othello. The program was able to learn from either source of information, but it was unclear whether or how one could learn simultaneously from both sources. We are in the process of finishing the integration of the methods in Othello, and are in the early stages of experimenting with an integrated approach in learning to control air traffic.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. We thank Rich Sutton, Andy Barto, Sharad Saxena, Jamie Callan, Tom Fawcett, Carla Brodley, and Margie Connell for helpful comments and discussion.

References

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley & Sons.

Huberman, B. J. (1968). *A program to play chess end games*. Doctoral dissertation, Department of Computer Sciences, Stanford University.

Lee, K. F., & Mahajan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence, 36*, 1-25.

Nilsson, N. J. (1965). *Learning machines*. New York: McGraw-Hill.

Samuel, A. (1963). Some studies in machine learning using the game of Checkers. In Feigenbaum & Feldman (Eds.), *Computers and Thought*. New York: McGraw-Hill.

Samuel, A. (1967). Some studies in machine learning using the game of Checkers II: Recent progress. *IBM Journal of Research and Development, 11*, 601-617.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*, 9-44.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In Touretzky (Ed.), *Advances in Neural Information Processing Systems*. Morgan Kaufmann.

Utgoff, P. E., & Saxena, S. (1987). Learning a preference predicate. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 115-121). Irvine, CA: Morgan Kaufmann.

Utgoff, P. E., & Heitman, P. S. (1988). Learning and generalizing move selection preferences. *Proceedings of the AAAI Symposium on Computer Game Playing* (pp. 36-40). Palo Alto, CA.