Why Conditional Approach is Hard to Learn

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QN
Email: Chris.Thornton@cogs.susx.ac.uk
Tel: (44)273 678856

December 14, 1994

CSRP 359

Abstract

The paper presents the results of an empirical study in which supervised learning algorithms were used to train an animat to perform a difficult navigation task. The results of the study are explained in terms of a theoretical distinction between two learning-complexity classes.

1 Introduction

Recently there has been increasing interest in the use of learning for the automatic acquisition of animat behaviors. Such behaviours are typically 'choicefull', which means that they entail (or allow for) a variety of responses in some or all situations. With choicefull behaviours it is convenient to use reinforcement learning regimes rather than supervised regimes, since without knowing each situation's correct response it is difficult to draw up a training set. With 'choiceless' behaviours, on the other hand, each situation does have a correct response. The derivation of a training set and implementation of a supervised regime is thus possible. In fact, a supervised regime applied to a choiceless behaviour should always perform at least as well as any reinforcement regime. The only difference between the supervised regime and the reinforcement regime (applied to a choiceless behaviour) is the fact that the feedback provided by the reinforcement regime is a *degraded* (e.g., noisy or intermittent) signal of the correctness of a given action/response. Thus for a reinforcement regime to outperform a supervised regime on a choiceless behaviour, it would have to be the case that the acquisition performance improves while the quality of the feedback signal degrades. This is clearly absurd.

The present paper looks at the learning of a choiceless behavior dubbed 'conditional-approach' by a variety of supervised methods. No reinforcement methods were examined on the assumption that they could not be expected to perform any better than the supervised methods. The behaviour was modeled in an animat with a simple sensory system and a motor system enabling forward and rotational moves. The behavior itself, which involves moving in on any small object in the sensory field but 'standing clear' of any large object, seems rather straightforward. However, it turns out to be poorly learned by supervised methods. We explain this 'failure-to-learn' using a statistical analysis.

The paper is divided into six main sections. This, the first section, is an introduction. In the second section we describe the comparative study, the simulation setup used, the training-data derivation method and the results obtained. In the third section we review basic methods for analyzing statistical properties of training sets and make a distinction between two types of generalization effect. In the fourth section we analyze statistical properties of training sets for the conditional-approach behavior and explain why it is hard to learn. In the fifth section we speculate on the form of general solutions to the conditional-approach learning problem. In the sixth and final section we offer a summary and some concluding comments.

2 The comparative study

The empirical basis of the paper is a comparative survey that investigated a behavior called 'conditional-approach'. The production of this behavior in an animat requires a proximity sensing system of some sort and motor abilities enabling forward and rotational movements. The behavior involves moving in on any relatively small object in the sensory field but standing clear of (i.e., *not* moving in on) any large object.

The behavior was investigated using computer simulations. The simulations used a 2-dimensional, rectangular world and a single animat. This had two free-wheeling castors situated fore and aft and two drive wheels situated along the central, latitudinal axis (see Figure 1). The animat was equipped with a range-finding system. This sensed the proximity of the nearest object — subject to 10% noise — along seven rays, evenly spaced within a 100 degree, forwards facing arc. A ray intersecting an object at point-blank range yielded the maximal input value (1) while a ray intersecting no object at all yielded the minimal input value (0).

The plan view shown in Figure 2 illustrates the basic simulation setup. The animat, situated in the lower part of the space, is represented as a small box with an arrow pointing in its direction of motion. The seven dashed lines are its probe rays. The boundaries of the space — here shown as unbroken lines — are actually transparent to the animat. Thus, in the situation shown, the animat senses only the circular blob directly ahead of it. That is to say, within its seven proximity inputs, the two associated with the rays intersecting the blob will be relatively high but the other five will be zeros indicating 'no object sensed'.

2.1 Control procedure

Within the training-set derivation process, the animat was driven using a set of four rules. These implicitly assumed that the environment would contain no more than one object at any one time. The animat controller had no internal state. Thus, the animat behavior in each time cycle was affected solely by the current inputs. The rules were as follows.

(1) If all proximity inputs are at their minimal values (indicating no object sensed) then swivel ten degrees to the right by driving the left wheel at 1/3 its maximum speed keeping the right wheel stationary.

(2) Otherwise, calculate the ratio between the apparent width (i.e. number of rays intersected) and the apparent proximity (i.e. maximum ray value) of the object in the sensory field.

(3) If this ratio exceeds a given threshold then stay still.

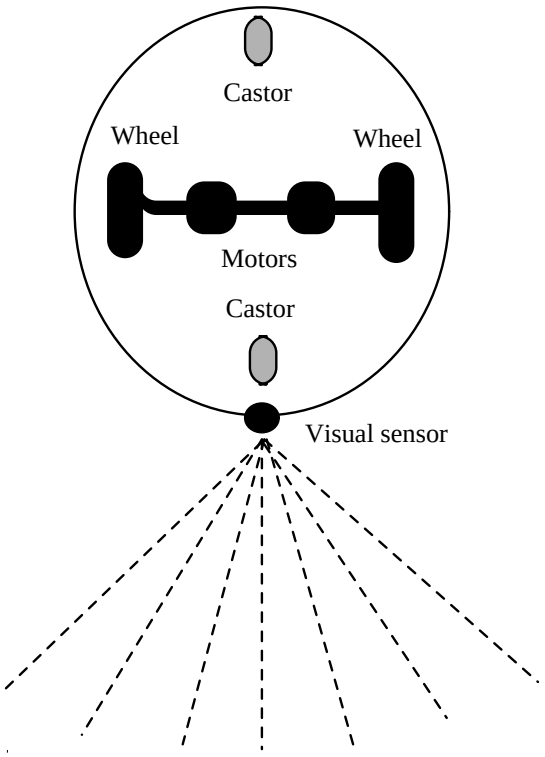

Figure 1: The simulated animat.

(4) Otherwise, move towards center of the object, by an amount equal to the length of the animat.

Figure 3 illustrates a short sequence within a typical simulation. Initially, the animat is situated in the lower part of the space. Its position corresponds to the lowermost rectangle in the figure. To begin with only the larger of the two round objects exists. The animat reacts to the presence of this object by moving forwards. (The series of rectangles show the sequence of positions occupied.) Once it has moved a little closer to the object, the width/closeness ratio exceeds the relevant threshold (see rule 2 above) and the animat halts. After a while the large object is removed and replaced with a smaller object slightly to the right. The animat responds by moving in on the small object.

2.2 Training the animat

The aim of the empirical investigation was to see how well supervised learning algorithms performed when used to train an animat to perform conditional-approach. To obtain training sets for the learning we repeatedly sampled the animat's reactions during simulation runs. This involved interrupting our simulation program in the middle of each time cycle and recording the sensory input received by the animat at that point, and the amount of drive being sent to the two wheels. The input/output pairs thus produced gave us the required training set.

The conditional-approach behavior entails producing three, basic behavioral responses to four scenarios.

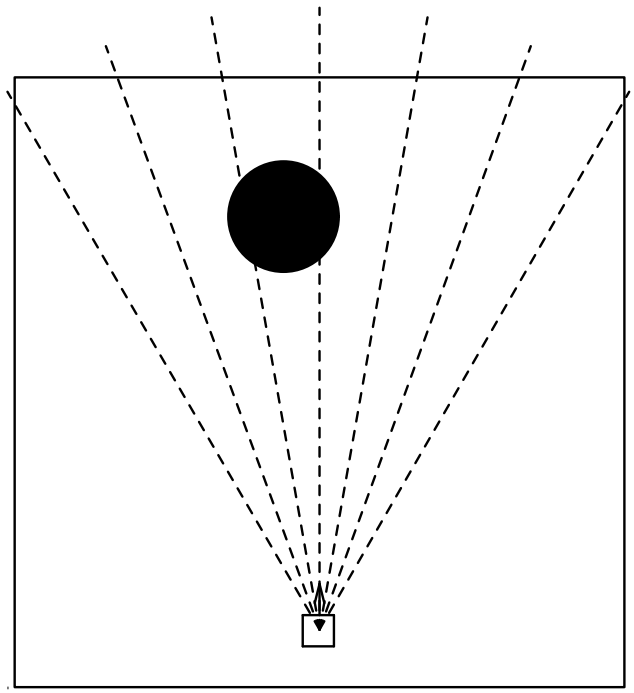

Figure 2: The simulation setup.

With no object appearing in the sensory field the animat must swivel right ten degrees. With an object appearing at long-range, or a *small* object appearing at close-range the animat must execute a forwards move towards that object. (This might or might not involve a change in direction.) With a large object appearing at close-range the animat should remain stationary.

To ensure that each of these responses had an equal representation within the training data we used the following initialization regime. Each time the animat arrived at a small object or remained stationary for more than 20 cycles, we reinitialized the environment, changing the size of the single object, and randomly choosing a new position for it. Thus, in each successive phase of the simulation, the animat would be confronted by an object of a different size and different relative position. The sampled stimulus-responses pairs thus contained roughly equal numbers of the four responses.

Our general strategy for testing the efficiency of training (with a particular learning algorithm) was as follows. Following derivation and presentation of the relevant training set (see above) we would re-run the simulation program interrupting it in the middle of each cycle. The animat's current proximity inputs would then be presented as a 'test case' to the relevant learning algorithm. The output returned would be used to drive the wheels of the animat. At the end of the simulation run, we would evaluate the overall behavior as a reproduction of the desired behavior.

2.3 Format of training examples

The inputs from the sensory system were represented (for purposes of training) in the form of real numbers in the range 0.0-1.0. The inputs formed a normalized measure of proximity and embodied

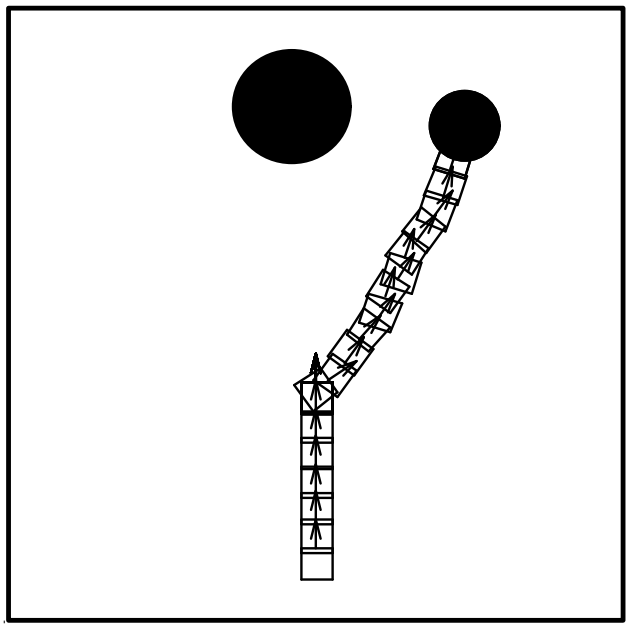

Figure 3: Conditional-approach behaviour.

10% noise. The amount of drive applied to the two wheels in each simulation step was represented in the form of two real numbers, also in the range 0.0-1.0. Thus, a full right turn with no forwards motion would appear in the training set as the pair <1.0,0.0> (given the assumption that the first number sets the drive on the left wheel and the second number the drive on the right wheel).

A sample of training pairs derived for the conditional-approach task is shown in Table 1. Note that the first seven numbers in each row (training pair) are the noisy proximity inputs. These are labeled v1, v2, v3 etc. The final two numbers in each row specify the required amount of drive to be applied to the two drive wheels. These are labeled d1 (amount of drive to the left wheel) and d2 (amount of drive to the right wheel). The first row shows a case of 'standing off' from a large object: the amount of drive for both wheels is 0.00. The second row illustrates the default behavioral response (swivel ten degrees to the right) produced whenever all the proximity inputs are zeros (indicating no object has been sensed). The swivel effect is achieved by setting the amount of drive for the right wheel to be 0.3.

2.4 Algorithms and parameter settings

The use of standard-format training sets enabled us to test the performance of any supervised learning algorithm on the conditional-approach problem. In practice we tested the performance of a wide range of algorithms including ID3 [1] and C4.5 [2], feed-forward network learning algorithms of the backpropagation family including 'vanilla' backpropagation [3], a second-order method based on conjugate-gradient descent [4] and a second-order method based on Newton's method called 'quickprop' [5]. We also tested a constructive network learning method called 'cascade-correlation' [5] and a classifier/genetic-algorithm combination based on Goldberg's 'simple classifier system' [6].

v1	v2	v3	v4	v5	v6	v7	d1	d2
0.00	0.00	0.00	0.27	0.38	0.33	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.3	0.00
0.00	0.00	0.81	0.81	0.81	0.79	0.78	0.00	0.00
0.00	0.87	0.88	0.89	0.89	0.89	0.00	1.00	1.00
0.00	0.00	0.00	0.27	0.38	0.33	0.00	0.00	0.00
0.73	0.74	0.00	0.00	0.00	0.00	0.00	0.4	1.00
0.81	0.81	0.81	0.79	0.78	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.85	0.84	0.82	0.00	1.00	0.80
0.00	0.00	0.00	0.78	0.78	0.00	0.00	1.00	1.00
0.00	0.00	0.00	0.00	0.76	0.77	0.76	1.00	0.80

Table 1:

The standard ID3 algorithm has no user-definable parameters. Thus there is only one way to apply it to a particular training problem. It produces as output a standard-format decision tree in which the leaf nodes are labeled with specific output cases and each internal node tests the value of a particular input variable. C4.5 is a more efficient version of ID3 that enables various parameters to be set to control tree-pruning actions. However, in all cases reported we used the program in unrestricted mode, i.e., with parameters set so that it would perform no pruning whatsoever.

All the network algorithms tested operate by modifying the connection weights in a fixed, non-recurrent network of artificial neurons (using the standard logistic activation function). The efficiency of network learning is determined by feeding in novel inputs to the network and seeing what outputs are generated after the activation has propagated across all the relevant connections. When applying network learning algorithms the user must decide the internal architecture of the network[1] and, in some cases, the learning and momentum rate. When testing the various network learning algorithms we experimented with a range of two-layered, feed-forward architectures (with complete inter-layer connectivity) but found that the best performance was obtained using nine hidden units; i.e. we settled on a 7-9-2 feed-forward architecture. All the results reported relate to this case.

When testing standard backpropagation we found that a learning rate of 0.5 and a momentum of 0.9 gave best results and these were the settings used in all the cases reported. When testing iterative learning algorithms (i.e., the network learning algorithms) we ran the algorithms for a minimum of 100,000 epochs of training (i.e., 100,000 complete sweeps through the entire training set).

In testing the classifier-system/genetic algorithm combination we used an implementation based closely on Goldberg's 'simple classifier system'. The classifier population was configured to include a 50/50 mixture of intermediate and final classifiers. That is to say, the actions for half the classifiers in any population were output patterns and the actions for the other half were input patterns. The standard bucket-brigade algorithm was used with standard defaults (e.g., as used in [6]). The codons in the input and output messages of the classifiers were single bytes encoding a real number in the range 0-1, with two decimal places of accuracy. The genetic algorithm used employed the crossover operator with random bit-mutation applying with a probability of 0.1. A fixed population size of 250 was used with 20% of the population being replaced by the genetic algorithm after every 5000 epochs (i.e., 5000 applications of the classifier system to the entire training set).

[1] The configuration of input and output units is fixed by the learning problem.

2.5 Results

The results can be roughly summarized by saying that C4.5 and nearest-neighbors performed better on the learning task than the connectionist algorithms or the classifier system, but that none of the algorithms provided satisfactory performance on this problem. In general, following training the animat would tend to either approach all objects (large or small) or no objects. It would only very occasionally produce the desired discrimination between large and small objects.

We measured the success of the training in several ways. First of all we measured conventional error rates (i.e. proportion of incorrect responses on unseens). However, these figures give a misleading impression of success. The majority of responses in the conditional-approach behavior do not entail making the crucial discrimination between large and small objects. They merely involve continuing rotatory behavior or moving further towards a small and/or distant object. A better performance measure is provided by sampling the frequencies with which the animat actually arrives at large and small objects. The former frequency we call the 'nip frequency', the latter the 'meal frequency'. These frequencies tend to show the extent to which the animat's behavior embodies the necessary size discrimination.

Our main results are summarized in Table 2. The figures in the 'hand-sim' row show the performance of the animat running under the control of the four rules shown above. The figures in the 'random' row show the performance obtained using a random move generator. The figures in the 'quickprop' row show performance after training with the 'quickprop' version [5] of the backpropagation algorithm [3]; the figures in the row labeled 'c4' show the performance after training with the C4.5 version of ID3 [2]; the figures in the row labeled 'NN' show performance after training with the nearest-neighbours algorithm [7]; finally, the figures in the row labeled CS show performance after training with the simple classifier system/genetic algorithm. All the figures are averaged over 10 different runs. The results reported were gathered using training sets containing 80 training examples since trial and error showed that this size of training set was sufficient to achieve negligably low error on the training examples from all of the algorithms tested.

	Error rate	Meal freq.	Nip freq.
hand-sim		0.864	0.090
random		0.014	0.043
quickprop	0.221	0.201	0.321
c4	0.233	0.479	0.371
NN	0.161	0.117	0.191
CS	0.344	0.251	0.275

Table 2:

The lowest error rate on the testing cases was 0.161 (16.1%) and this was produced by the nearest-neighbours algorithm (NN). This figure seems low but actually reveals relatively poor performance (for reasons explained above). The same goes for the other error rates shown. The columns headed 'Meal freq.' and 'Nip freq.' show the 'meal' and 'nip' frequencies respectively for the various simulated animats. Note that the hand-coded animat achieves a high meal-to-nip ratio while the trained animats do quite poorly, with the quickprop, NN and CS animats achieving nip-frequencies in excess of the meal-frequencies. As is to be expected, the randomly driven animat also achieves a nip-frequency in excess of its meal-frequency since the probability of randomly bumping into a large object is larger than the probability of randomly bumping into a small object.

2.6 Comparison with other behaviors (obstacle-avoidance and pursuit)

In view of the possibility that the poor performance obtained from the learning algorithms was due to some flaw in the overall methodology, we carried out some additional experiments. These aimed to discover if we could use the same algorithms and the same methodology to learn more familiar animat behaviors. In particular, we tested the learning algorithms on 'obstacle-avoidance' and 'pursuit'.

For these experiments we used the parameter settings for the learning algorithms described above except in the case of network learning algorithms applied to the obstacle-avoidance task, where we used feed-forward architectures containing just two hidden units with complete connectivity between layers. We also used the same simulation setup but with a modified animat in the case of the obstacle-avoidance training. This animat had the usual two-wheel drive system but it used a simplified sensory system embodying just two proximity probes arranged in a 10 degree, front-facing arc (i.e., it had one probe ray offset five degrees on each side of the forwards direction). The environment for the obstacle-avoidance training was also modified so as to contain three, oval or rectangular objects. The environment was also configured so that the boundaries of the space appeared opaque to the animat. Thus, the simulated animat was able to 'see' both the edges of the objects and the edges of the world.

The control procedure for the obstacle-avoidance simulations was as follows.

(1) Find the higher of the two proximity inputs.

(2) If this value exceeds 0.8 then swivel ten degrees to the right.

(3) Otherwise move forwards by an amount proportional to the length of the animat.

In Figure 4 we see a short trace of a simulated animat producing obstacle-avoidance behavior. The animat's position in each simulation step is shown as a small, arrow-topped box as before. Thus the sequence of boxes shows the animat's trajectory around the environment. Note how the trajectory steers clear of all the obstacles and the boundaries of the space.

2.6.1 Pursuit

The second behavior examined was 'pursuit'. For this behavior we used exactly the same experimental setup as for conditional-approach; i.e., we used a simulated animat with seven probe rays arranged in a 100-degree arc. The environment contained no objects and its boundaries were transparent.

Within the training simulation, the animat tracked a second simulated animat. The shape of this second animat was rectangular and its size was arranged such that it would just intersect two of the seven probe rays at 75% of the maximum animat-to-animat distance. The second animat (henceforth the 'leading animat') moved around the environment according to the following probabilistic regime. In each cycle, there was a probability of 0.3 of the leading animat moving forwards, a probability of 0.35 of it making a forwards+left move and a probability of 0.35 of it making a forwards+right turn. The step size for the leading animat (i.e., the total amount of drive that could be applied to the wheels) was arranged to be 125% that of the pursuing animat. Thus the leading animat had a small speed advantage over the pursuing animat. In Figure 5 we see a trace of a simulated animat producing the pursuit behavior. The pursuing animat is shown here using dashed lines. The leading animat is shown using unbroken lines.

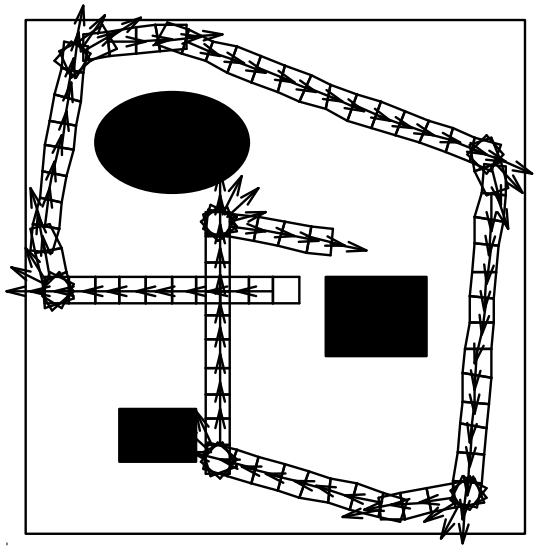

Figure 4:

2.7 Results for obstacle-avoidance and pursuit

The results for this second phase of experiments can be summarized by saying that *all* the learning algorithms looked at were able to learn the two behaviors rather easily. The main performance figures are shown in Table 3. The column headed 'avoidance-CF' shows the crash frequencies for the various animats performing obstacle-avoidance while the column headed 'pursuit-AD' shows the 'average distance to target' for the animats executing the pursuit behavior. (The distances are proportional to the size of the space.) The row labels are the names of the relevant learning algorithms as before. Note that the crash frequencies and the average distances for all the trained animats are low when compared with the randomly moving animat.

	avoidance-CF	pursuit-AD
hand-sim	0.000	0.048
random	0.780	0.246
conjgrad	0.006	0.076
ID3	0.006	0.041
NN	0.002	0.081
CS	0.009	0.088

Table 3:

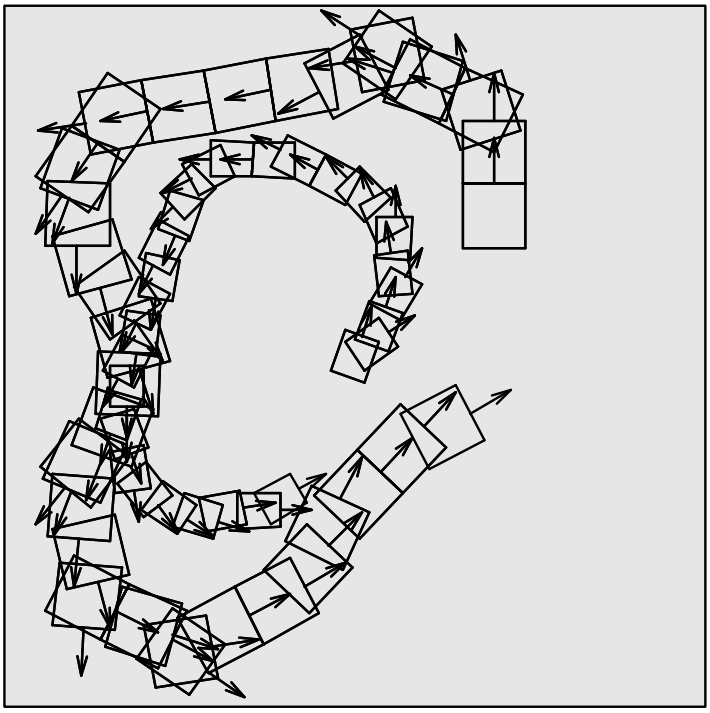

Figure 5:

2.8 Why is conditional-approach hard?

Our results with respect to obstacle-avoidance and pursuit are as per expectation. Behaviors of this type are known to be learnable by a variety of methods [8; 9; 10]. But the fact that we were able to obtain successful training using our simulation-based methodology shows that the failure of the algorithms to deal satisfactorily with conditional-approach is not necessarily due to a flaw in the basic methodology. The failure of the network learners on conditional-approach thus begins to seem increasingly paradoxical, especially in view of the fact that as a behavior it appears to be little more than an amalgam of obstacle-avoidance and pursuit.

To explain our results we will develop an analysis of the three behaviors. This will involve looking at the statistical properties of the training sets generated from the simulations. It will show that the conditional-approach behavior differs from obstacle-avoidance and pursuit in that its underlying rule is *not* strongly represented within the statistical regularities of a typical, simulation-derived, training set. As we will show, this tends to have the effect of making conditional-approach hard to learn by methods that rely primarily on exploiting statistical regularities.

3 Learning as the exploitation of justification

The process of learning implemented by some arbitrary learner mechanism can be conceptualized as the acquisition of a target input/output mapping.[2] To have any chance of success the learner requires some source of feedback regarding the mapping to be acquired. In the much studied supervised learning scenario, this feedback takes the form of a set of examples taken from the target mapping. The learner's aim is to arrive at the point at which it is able to map any input taken from the mapping onto its associated output. In more general terms, the learner's aim is to be able to give a high probability to the correct output for an arbitrary input taken from the mapping.

If the learner is to have any chance of achieving this goal, the feedback it receives must contain information which justifies the assigning of particular probabilities to particular outputs. Thus we see that learning is essentially the process of discovering and exploiting such justifications. To understand the nature of the process we need to analyze the ways in which supervisory feedback can provide justifications for assignments of particular probabilities to particular outputs.

There are two main cases to consider. Supervisory feedback (i.e., sets of examples) can justify probability assignments either *directly* or *indirectly*. Direct justification is provided if the probabilities in question are observed directly within the training examples (in the form of frequencies). They are justified indirectly if they can be derived from those examples, either by a recoding of the original examples or by some other analytic process.

The distinction between the direct and indirect forms of justification can be illustrated with an example. Consider the following training set. This is based on two input variables (x1 and x2) and one output variable (y1). There are six training examples in all. They are laid out with one example per line. An arrow separates the input part of the example from the output part. The values of the two input variables appear on the left of the arrow. The value of the output variable appears on the right.

```
x1  x2      y1

1   2  -->  1
2   2  -->  0
3   2  -->  1
3   1  -->  0
2   1  -->  1
1   1  -->  0
```

A wide variety of probabilities can be observed in these training examples. To begin with we have the probabilities for first-order cases (i.e., instantiations of a single variable). These are shown in Table 4. The 'C' column shows the case in question and the 'P(C)' column shows the observed probability of that case.

We can also observe the probabilities of many second-order cases — cases involving the instantiation of two variables. These are shown in Table 5.

Note that the probabilities for the third-order cases (i.e., the cases that specify values for all three variables) are degenerate. Assuming there is no duplication in the training data, each third-order case occurs exactly once. Thus its probability is necessarily 1/n where n is the size of the training set.

The probabilities introduced so far are all unconditional. *Conditional* probabilities can also be observed.

[2] The mapping for a choicefull behaviour is one-to-many.

C	P(C)
	1
x2=2	0.5
x2=1	0.5
y1=1	0.5
y1=0	0.5
x1=3	0.33
x1=2	0.33
x1=1	0.33

Table 4:

C	P(C)
x2=2, y1=1	0.33
x2=1, y1=0	0.33
x1=3, x2=2	0.17
x1=2, x2=2	0.17
x1=3, y1=1	0.17
x1=3, x2=1	0.17
x1=1, x2=2	0.17
x1=2, y1=1	0.17
x1=2, x2=1	0.17
x2=1, y1=1	0.17
x1=1, y1=1	0.17
x1=1, x2=1	0.17

Table 5:

For example, we can observe the conditional probability of observing a particular instantiation of the output variable for given first-order cases of the input variables. These probabilities are shown in Table 6. By the argument used previously, the second-order conditional probabilities here are degenerate since there is necessarily exactly one occurrence of each second-order case of the constrained variables.

In the supervised learning scenario, it is, of course, the conditional and unconditional probabilities relating to the *output variable(s)* which are of interest. Thus, Table 6 in conjunction with the table listing the first-order unconditional probabilities for the output variable, provide an exhaustive enumeration of all directly observed, probability-assignment justifications. A quick perusal of the two tables shows that none of the output-variable probabilities are particularly extreme. Thus the training examples do not contain directly observable justification for strong output predictions (i.e., predictions made with a probability close to 1).

What, then, of the indirectly observed justifications? These are obtained via recoding or analysis of the original examples. Imagine that we recode our training examples by substituting, in each training pair, the original input variables with a single variable whose value is just the difference between the original variables. This gives us a set of derived pairs as shown below (the value of x4 here is the difference between the values of x1 and x2).

C	P(C)	P(y1=0|C)	P(y1=1|C)
	1	0.5	0.5
x2=2	0.5	0.33	0.67
x2=1	0.5	0.67	0.33
x1=3	0.33	0.5	0.5
x1=2	0.33	0.5	0.5
x1=1	0.33	0.5	0.5

Table 6:

```
Original pairs              Derived pairs (x4 = |x1-x2|)

x1  x2       y1             x4       y1

1   2  -->   1              1  -->   1
2   2  -->   0              0  -->   0
3   2  -->   1              1  -->   1
3   1  -->   0              2  -->   0
2   1  -->   1              1  -->   1
1   1  -->   0              0  -->   0
```

The probabilities we *directly* observe in these derived training data are, by definition, indirectly observed probabilities in the original training data. However, they are still probabilities. Thus we can derive tables of conditional and unconditional probabilities in the usual way. The unconditional and conditional output probabilities for the derived training set are shown in Table 7.

C	P(C)	P(y1=0|C)	P(y1=1|C)
	1	0.5	0.5
x4=1	0.5	0.0	1.0
x4=0	0.33	1.0	0.0
x4=2	0.17	1.0	0.0

Table 7:

Note how the recoding we applied to the training examples has produced data in which we observe a number of extreme probabilities relating to the output variable y1. The recoding thus provides us with indirect justification for predicting y1=0 with a probability of 1, if the difference between the input variables is 1. It also provides us with indirect justification for predicting y1=1 with a probability of 1, if the difference between the input variables is either 2 or 0. In short, we have indirect justification for the output rule 'y1=1 if x4=1; otherwise y1=0'.

4 Complexity implications for type-1 problems

As I noted above, the aim in supervised learning is to be able to identify target outputs with high probability. We have now seen how the *justifications* for such assignments contained within the learner feedback (i.e., training examples) are either directly or indirectly observed. Discovering direct forms involves deriving probability statistics. Discovering indirect forms involves (1) deriving a recoding of the training examples and (2) deriving probability statistics within the recoded data.

From this we can draw a preliminary conclusion regarding the generic complexity of learning problems. Finding a solution to a particular learning problem necessarily entails discovering some combination of two forms of justification. The number of direct (henceforth 'type-1') justifications is exponentially related to the number of variables and values involved in the training examples. Thus it is typically large but finite.

The number of indirect (henceforth 'type-2') justifications is the number of direct justifications (derivable from the relevant recoded data) plus the number of possible recodings of the data. The number of possible recodings is simply the number of distinct Turing machines we can apply to those data. There are infinitely many of these. Thus the space of indirect justifications is infinitely large.

The task of discovering justifications is naturally viewed as a search. In the case of direct justifications the search takes place in a finite space while in the case of indirect justifications the search takes place in a space that is infinitely large. Thus, other things being equal, learning problems which necessitate the discovery of indirect justifications are 'harder' than problems which do not do so. This is the first and most fundamental observation to be made regarding generic learning-problem complexity.

Unfortunately, translating the observation into a computable measure of learning-problem complexity is not straightforward. The difficulty of a learning problem with a solution based on type-1 justifications (henceforth a 'type-1 problem') can be measured in terms of the difficulty of discovering the relevant type-1 justifications. This is a function of (1) the number of justifications involved and (2) the size of the space in which they exist.[3] It is a straightforward matter to calculate the size of the space. But counting the number of type-1 justifications required involves working out *what they are*. Thus, measuring the degree of difficulty of a type-1 learning problem requires *solving* that problem (i.e., finding its type-1 solution).

This is not necessarily a disadvantage. Even when the search for type-1 effects is executed exhaustively, it may still be tractable. But, in fact, by using learning algorithms which perform a close approximation of the type-1 discovery task, we can effectively measure type-1 complexity much more efficiently than is done via exhaustive search.

The well-known ID3 algorithm [1] is perhaps the most obvious candidate for our purposes here.[4] This supervised algorithm builds a solution by recursively splitting the training cases into subsets until the point is reached where each subset is associated with a unique output. The splits are arrived at by dividing up the relevant set of cases according to their value on a particular input variable. Target outputs are generated by finding which output is associated with the leaf node accessed by the relevant input.

The efficiency of the algorithm derives from the fact that, at each state, a split is selected which maximizes the output uniformity (minimizes the entropy) of the subsets produced. In the initial case — where a split is sought for the complete training set — the process is effectively identifying the input variable which correlates most strongly with the outputs. This variable is the one which participates

[3] There is no commitment in this to the idea that the learning method actually carries out a search.
[4] See also the latest variant of ID3 called C4.5 [2].

in the most pronounced conditional probability effects with the output variable(s). ID3's initial aim, therefore, is to identify the input variable which participates in the most informative set of first-order, type-1 justifications. In cases where these justifications provide a satisfactory solution to the problem, the algorithm is guaranteed to find the ideal type-1 solution.

If the type-1 solution involves higher-order probability effects then ID3 is not guaranteed to find the ideal type-1 solution. But in many cases this does not diminish the algorithm's suitability as an approximate type-1 discovery method. Supervised learning problems are typically prepared so as to ensure statistical independence of input variables. If this property is obtained, then higher-order justifications will not exist and any type-1 solution will necessarily be the first-order solution identified by ID3.

Various other learning algorithms exist which provide effective methods for accessing type-1 justifications. Such methods are often biased towards first-order justifications (cf. the Least-Mean-Squares [11] and Perceptron [12] methods) and are thus subject to the same reservations — and recommendations — as the ID3 algorithm. Methods related to backpropagation [3], which do not have such an obvious first-order bias, can potentially be used to find higher-order type-1 solutions (se further discussion below).[5]

5 Complexity implications for type-2 problems

Type-2 justifications are probability effects which are only brought to light via a recoding of the original training data. (Intuitively, they are effects which are discovered via an 'analytic' process.) The fact that there are typically infinitely many possible recodings that might be applied to any given training set means that measuring the difficulty of a type-2 problem (a problem whose solution is based purely on type-2 effects) is impossible unless we set constraints on the nature of the recoding that can be applied.

Before looking at what this means, I will make some preliminary observations about the nature of type-2 effects. Recall that a type-1 effect is an output probability which is either unconditional or conditional on the absolute state of one or more input variables. A type-2 effect is an output probability which is 'associated' in some way with the states of input variables. But it cannot depend in any way on the *absolute* states of input variables since this would make it a type-1 effect! Thus it must be associated with the *relative* states of input variables. Putting it more simply, a type-2 effect must be based on a relational property of the input variables.[6]

This provides us with a rule-of-thumb for deciding whether a learning problem has a type-1 or a type-2 solution. If the problem is based on a relational input/output rule, then probability effects affecting the output variable(s) cannot be based on absolute values of the input variables. Thus the solution can be expected to be comprised of type-2 effects. If the problem is based on a non-relational input/output rule, then the probability effects applying to the output variable must be based on absolute values of the input variables. Thus the solution can be expected to be comprised of type-1 effects.

Of course, in reality, nothing is this simple. Learning problems which are based on relational input/output rules typically have training sets which exhibit 'spurious' type-1 effects. The training set

[5] The Perceptron learning algorithm is, of course, a method which can only be successfully applied to linearly separable problems, i.e., discrimination problems which can be solved by identifying a linear hyperplane separating the relevant classes. Where the hyperplane is aligned with one of the axes of the input space, there will be marked correlations between values of the corresponding input variable and the output variable, and thus pronounced type-1 effects. However, where the hyperplane is unaligned such effects may disappear.

[6] In fact this is not quite true. The argument stated allows it to be based on a non-relational property of a single input variable provided that property effectively masks any type-1 effect.

shown above is a case in point. The input/output rule is relational and yet the type-1 analysis reveals that the training set does exhibit type-1 effects affecting the output variable.

Be this as it may, the basic rule-of-thumb for classifying learning problems does give us a possible explanation for the apparent difficulty of conditional-approach. The input/output rule for conditional-approach is clearly a relational one: it is, after all, based on the calculation of a *ratio* between two values, namely the apparent width and the apparent closeness of the object.[7] The rules for obstacle-avoidance and pursuit, on the other hand, are both based on establishing a direct correspondence between the apparent closeness of an object in the sensory field and a particular behavioral response. In the case of pursuit, the correspondence is a simple matter of sensory stimulation being transformed into drive-wheel activity: objects appearing in particular parts of the sensory field cause particular amounts of drive to be applied to the wheels to ensure an appropriate move/turn. In the case of obstacle-avoidance the correspondence is a matter of sensory stimulation indicative of very near objects inhibiting drive-wheel altogether.

Thus we have a fairly clear distinction between conditional approach on the one hand, and pursuit/obstacle-avoidance on the other. In conditional approach, but *not* in pursuit/obstacle-avoidance, the input/output rule is based on a relational property of the input variables. Any type-1 effects observed in training data for conditional-approach are effectively artifacts, i.e., purely accidental, statistical features attributable to the size/dimensionality/grain etc. of the experimental situation. These artifactual effects may be relatively intense — the complexity of the training set means that measuring them directly is out of the question — but the fact that supervised learners tend to perform badly on this problem, but very well on obstacle-avoidance and pursuit tends to suggest that conditional-approach produces training sets in which the type-1 effects are relatively weak.

6 Summary

The paper presented the results of a comparative study that looked at the conditional-approach behavior. The results of the study showed that several, powerful learning methods are unable to successfully learn the conditional-approach behavior even though they are perfectly capable of learning other, closely related behaviors such as obstacle-avoidance and pursuit.

The failure on conditional-approach training was explained using a statistical analysis. This showed that supervised learning involves generating probability assignments and that these may be justified by training data either directly or indirectly. Directly observed justification were labeled type-1 effects. Indirectly observed justifications were labeled type-2 effects. The latter were shown to exist in an infinitely large space and thus to be 'harder' to find. It was then shown that training sets for learning problems based on relational input/output rules will tend to exhibit type-2 justifications only and thus be harder to learn and the hardness of conditional-approach was explained by showing it to be based on a relational input/output rule.

References

[1] Quinlan, J. (1986). Induction of decision trees. *Machine Learning, 1* (pp. 81-106).

[2] Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. San Mateo, California: Morgan Kaufmann.

[7] In fact, width is, itself, a relativistic property of the inputs.

[3] Rumelhart, D., Hinton, G. and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323* (pp. 533-6).

[4] Becker, S. and Cun, Y. (1988). Improving the convergence of back-propagation learning with second-order methods. CRG-TR-88-5, University of Toronto Connectionist Research Group.

[5] Fahlman, S. and Lebiere, C. (1990). *The Cascade-Correlation Learning Architecture.* CMU-CS-90-100, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

[6] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

[7] Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis.* New York: Wiley.

[8] Nehmzow, U., Smithers, T. and Hallam, J. (1989). Really useful robots. In T. Kanade, F. Green and L. Hertzberger (Eds.), *Proceedings of IAS2, Intelligent Autonomous Systems* (pp. 284-292). Amsterdam.

[9] Cliff, D., Husbands, P. and Harvey, I. (1993). Evolving visually guided robots. In J. Meyer, H. Roitblat and S. Wilson (Eds.), *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive Behaviour* (SAB92). MIT/Bradford Books.

[10] Millan, J. (forthcoming). On autonomous mobile robots and reinforcement connectionist learning. *Neural Networks and a New AI.* Chapman and Hall.

[11] Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence*, *40* (pp. 185-234).

[12] Minsky, M. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry* (expanded edn). Cambridge, Mass.: MIT Press.