Supervised Learning of Conditional Approach: A Case Study

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QN
Email: Chris.Thornton@cogs.susx.ac.uk

November 25, 1993

Abstract

Reinforcement learning regimes have been shown to be capable of learning animat behaviours such as 'obstacle avoidance' and 'wall following'. Such behaviours can usually be learned more quickly using ordinary supervised methods, since in this case the learner receives more direct feedback. However, 'conditional approach' behaviour (move in on small objects but stand clear of large ones) seems to be hard to learn even by neural network learning methods such as backpropagation. The paper presents the results of a study which investigated this behaviour and shows how the 'hardness' of the behaviour can be accounted for in statistical terms.

1 Introduction

Recently there has been increasing interest in the use of learning for the automatic acquisition of animat behaviors (e.g., [1,2]). Attention usually focusses on reinforcement methods. However, ordinary supervised methods can be used provided there is some method available for generating suitable training sets. These can be expected to produce better performance in general since (1) they receive more informative feedback and (2) they are not required to solve a credit-assignment problem in addition to the learning problem.

The present paper looks at the learning of a behavior dubbed 'conditional-approach' by supervised methods. This behaviour is 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 based on a qualitative distinction between two classes of generalization effect.

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'.[1] 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.[2] 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

[1] A colour video showing the simulations performed and results obtained is available from the author.

[2] The simulations were run under the Poplog environment [3] running on a Sun SPARCstation 1+.

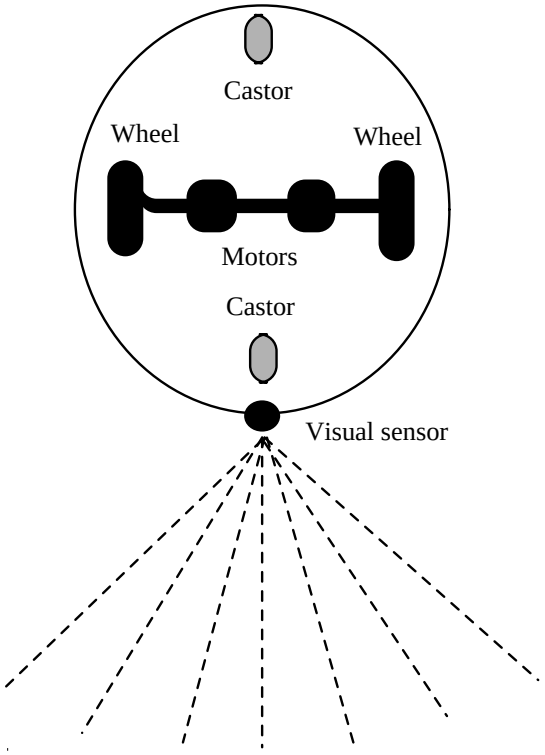

Figure 1: The simulated animat.

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 'sees' 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 at their lowest values, 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

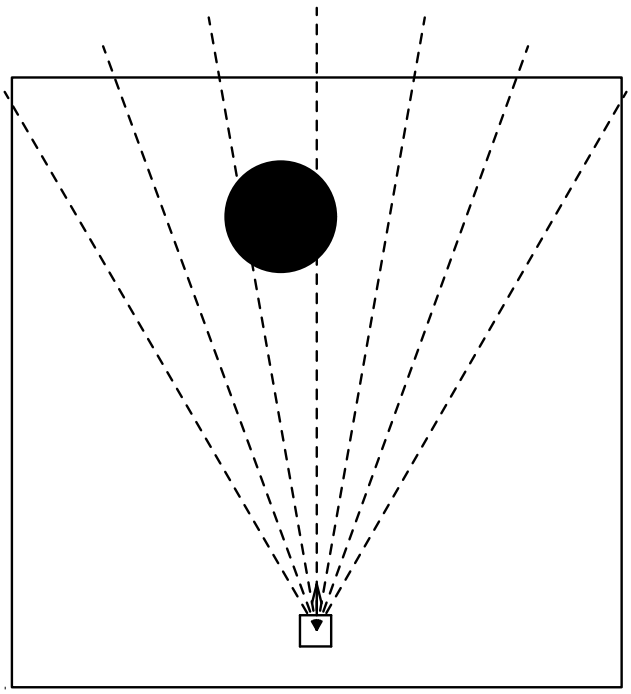

Figure 2: The simulation setup.

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. (The swivel was effected by driving the left wheel at 1/3 its maximum speed and 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.

(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.

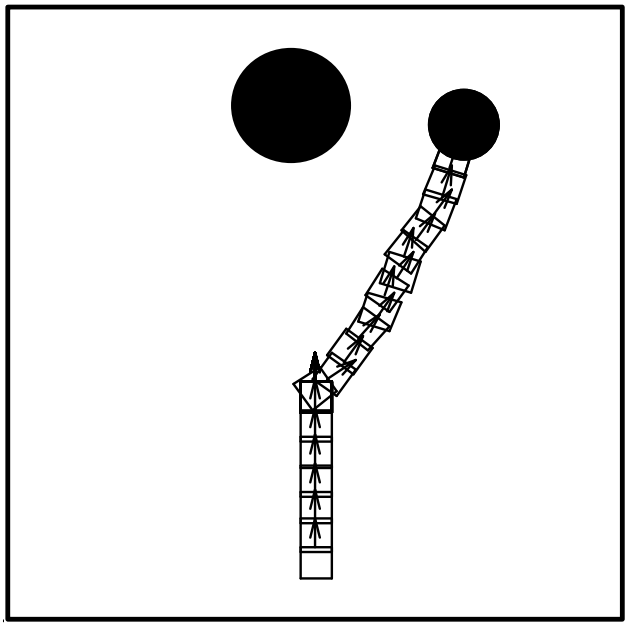

Figure 3: Conditional-approach behaviour.

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. Reinforcement learning methods are often used for purposes of animat training. However, we felt that such methods were not suitable for this study since they use restricted feedback (no explicit outputs are presented) and in most cases involve a credit-assignment problem in addition to the learning problem. The use of supervised algorithms requires the provision of a training

set of examples. To obtain these, 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. With no object appearing in the sensory field the animat must swivel rightwards by 10 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 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.

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:

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 [4] and C4.5 [5], feed-forward network learning algorithms of the backpropagation family including 'vanilla' backpropagation [6], a second-order method based on conjugate-gradient descent [7] and a second-order method based on Newton's method called 'quickprop' [8]. We also tested a constructive network learning method called 'cascade-correlation' [8] and a classifier/genetic-algorithm combination based on Goldberg's 'simple classifier system' [9].

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[3] 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 [9]). 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).

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

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

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 [8] of the backpropagation algorithm [6]; the figures in the row labeled 'c4' show the performance after training with the C4.5 version of ID3 [5]; the figures in the row labeled 'NN' show performance after training with the nearest-neighbours algorithm [10]; 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 pro-

duced 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.

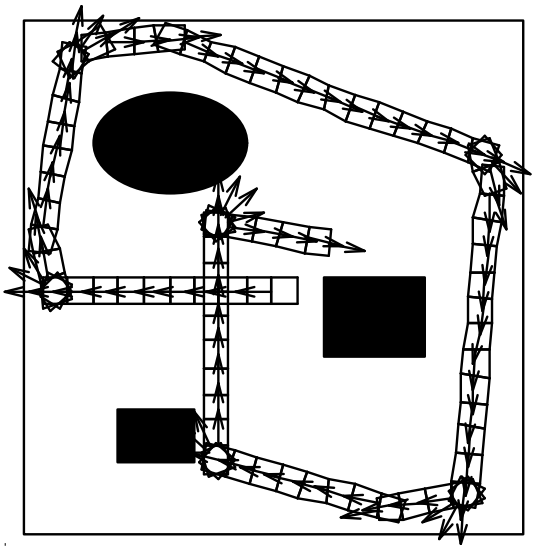

Figure 4:

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.

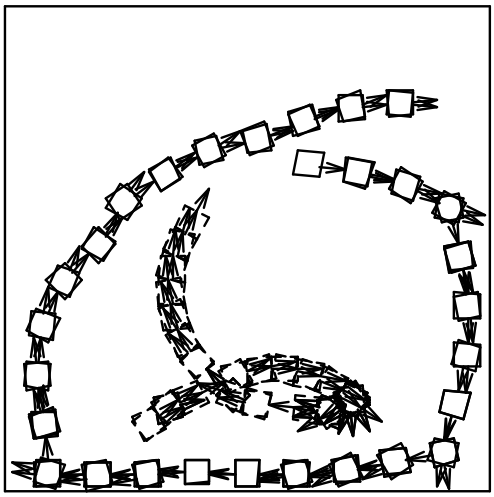

Figure 5:

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:

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 [11; 1; 12]. 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 seems 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 Statistical properties of training sets

Let us first make some general observations about the statistical properties of training sets. Consider the example training set shown below. This is based on two input variables (x1 and x2) and one output variable (x3). There are six training pairs in all. The pairs are laid out with one pair per line. An arrow separates the 'input vector' of the pair from the 'output vector'. 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      x3

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

In this dataset we can observe a number of instantiation n-tuples, henceforth called *cases*. First-order cases are instantiation 1-tuples. Examples include <x1=3>, <x3=0> and <x2=2>. Second-order cases are instantiation 2-tuples. An example is <x1=3, x2=1>. This case is observed in the fourth line of the training set. A second-order case from the second line of the training set is <x3=0, x1=2>. Since there are only three variables in all there is exactly one third-order case for each line (i.e., instantiation vector) of the training set.

Given a particular case, we can compute the frequency with which it appears in the training set. The frequencies for all first and second-order cases in the training data above are shown in Table 4. Note that the frequencies 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 frequency is necessarily 1/n where n is the size of the training set.

3.1 Conditional frequencies

The frequencies shown in Table 4 are *unconditional* frequencies. We can also derive the *conditional frequencies*.[4] These are frequencies that exist with respect to a particular constraint over variable instantiations. In Table 5 we see the frequencies for particular instantiations of the output variable (x3) given possible constraints on other variables. (As before, the column headed 'Freq' shows the absolute frequency for the main case.)

By the argument used previously, the 2nd-order conditional frequencies here are of no interest since there is necessarily exactly one occurrence of each 2nd-order case of the constrained variables.

[4] These can be construed as Bayesian probabilities [10].

Case	Freq.
	1
x2=2	0.5
x2=1	0.5
x3=1	0.5
x3=0	0.5
x1=3	0.33
x1=2	0.33
x1=1	0.33
x2=2 + x3=1	0.33
x2=1 + x3=0	0.33
x1=3 + x2=2	0.17
x1=2 + x2=2	0.17
x1=1 + x2=2	0.17
x1=3 + x2=1	0.17
x1=3 + x3=1	0.17
x1=2 + x2=1	0.17
x1=2 + x3=1	0.17
x2=1 + x3=1	0.17
x1=3 + x3=0	0.17
x1=1 + x3=1	0.17

Table 4:

3.2 Type-1 versus type-2 frequencies

A clear distinction must be made between cases (such as those considered above) that can be observed *directly* in the training data, and cases that can only be observed *indirectly*. For our purposes, a case can be observed indirectly if it can be observed directly in some systematic *recoding* of the original data. What this means is that an instantiation n-tuple that occurs in some reformulation of the original data, is considered to be a case that is 'observed indirectly' in the *original* data. We will call frequencies for directly observed cases type-1 or *empirical* frequencies.[5] We will call frequencies for indirectly observed cases type-2 or *mediated* frequencies.

The difference between the two types of frequency can be illustrated by recoding our original training set. Imagine that we reformulate the inputs (from above) by substituting — in each training pair — the original input variables with a single

[5] We call them 'empirical' to try to emphasize the fact that they are, in some sense, 'in' the data.

Constraint	Freq.	Fr. x3=0	Fr. x3=1
	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 5:

variable whose value is just the difference between the original variables. This gives us a set of derived pairs as shown in Figure 6 (the value of x4 here is the difference between the values of x1 and x2). The frequencies we *directly* observe

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

x1  x2      x3              x4      x3

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
```

Figure 6: Recoding of training set.

in this derived training set are type-2 (mediated) frequencies with respect to the original training data. However, they are still frequencies. Thus we can derive tables of conditional and unconditional frequency statistics in the usual way. The unconditional frequencies for the derived training set are shown in Table 6.

The conditional frequencies for instantiations of x3 given instantiations of x4 are shown in Table 7.

Constraint	Freq.
	1
x3=0	0.5
x3=1	0.5
x4=1	0.5
x4=0	0.33
x4=2	0.17
x4=1 + x3=1	0.5
x4=0 + x3=0	0.33
x4=2 + x3=0	0.17

Table 6:

Constraint	Freq.	Fr. x3=0	Fr. x3=1
	1	0.5	0.5
x4=0	0.33	1.0	0.0
x4=2	0.17	1.0	0.0
x4=1	0.5	0.0	1.0

Table 7:

3.3 Classes of regularity

For the purposes of empirical learning the statistical regularities in the training data are of central importance. But our perception of these is determined by our assumptions with respect to *chance*. The 'chance-level' for a frequency is simply the frequency (or in general range of frequencies) that we expect to see given purely random effects. Where we observe a frequency that diverges markedly from what we assume to be its chance level(s) we necessarily believe that non-random processes are at work. For present purposes we will assume that the chance-level for an instantiation of a random variable capable of taking n values is precisely 1/n, no more and no less. This is a 'least-conservative' approach since we are assuming that any frequency that diverges from 1/n, no matter by how small an amount, is to count as a non-chance-level frequency. (Since we aim to make a purely theoretical point this lack of conservatism is of no consequence.)

A non-chance-level frequency is caused (by assumption) by a non-random effect and thus forms a statistical regularity. Since we have distinguished two types of frequency effect, we can distinguish two types of regularity.

- **Type-1 regularity**: divergence from chance-levels in type-1 frequencies.
- **Type-2 regularity**: divergence from chance-levels in type-2 frequencies.

To place this in a concrete setting, consider the example training sets shown above. The output variable x3 is a binary variable. Thus the frequency for either of its two possible instantiations is exactly 0.5. When we look at the type-1 conditional frequencies for the training data we see that most of the values are at or close to their chance-level of 0.5. However, when we derive relevant the type-2 conditional frequencies (after recoding in the suggested way) we obtain a frequency table in which *every* value diverges *maximally* from its chance level. Intuitively, then, the training set can be classified as exhibiting more type-2 regularity than type-1.[6]

The frequency effects brought to light by the recoding translate naturally into a completely general input/output rule. The table of type-2 conditional frequencies makes it obvious that x3=1 if and only if x4=1. From this we trivially obtain the input/output rule 'x3=1 if x4=1; otherwise x3=0.' Thus we see how the recoding effectively brings the regularity underlying the training set to the surface. Once this has happened it is a straightforward matter for a learning algorithm to exploit it. Recognizing the strong, mutual interdependence between learning and regularity leads us to distinguish three classes of learning problem.

- **Pure type-1 learning problems**: problems that involve exploiting type-1 regularities only,
- **Pure type-2 learning problems**: problems that involve exploiting type-2 regularities only, and
- **Hybrid problems**: problems that involve exploiting some mixture of both types.

3.4 Parity problems are pure type-2

The distinction between type-1 and type-2 problems is nicely illustrated by the so-called 'parity' problems (cf. [13]). Complete parity mappings show no type-1 regularity at all. Their empirical frequencies are always *exactly* at their chance levels. The input/output rule for a parity mapping is simply that the output should be 1 (or true) just in case the input vector contains an odd number of 1s (or, in general, an odd number of odd values). The complete mapping for the third-order, binary-valued parity problem (i.e., 3-bit parity) is as follows.

[6] The question of how type-1 regularity should be measured formally is addressed below.

x1	x2	x3		x4
1	1	1	-->	1
1	1	0	-->	0
1	0	1	-->	0
1	0	0	-->	1
0	1	1	-->	0
0	1	0	-->	1
0	0	1	-->	1
0	0	0	-->	0

Every single first and second-order conditional frequency for this mapping (for values of the output variable x4) is at its chance level of 0.5. And, in fact, the frequency statistics for parity mappings are *always* like this. If we are dealing with n-bit parity then the highest order, non-degenerate frequencies are the (n-1)th-order frequencies. Given binary variables we will necessarily find exactly two occurrences of each (n-1)th-order case in the training set, and these two cases will necessarily show a different value for the 'other' variable. Thus the conditional frequencies for the case in question will be evenly distributed between the different output cases and the conditional frequencies for instantiations of the output variable will always be identical. If they are identical, they must be at their chance level. Thus, parity problems are always pure type-2.[7]

3.5 Complexity implications

Distinguishing between type-1 and type-2 problems helps to shed light on the complexity implications of different learning scenarios. Type-2 regularities are non-chance frequencies for cases observed in some *reformulation* of the original data. Thus, points in the space of type-2 regularities correspond to possible data reformulations, i.e., possible computational devices (Turing machines) capable of processing those original data. The space of possible type-1 regularities, on the other hand, is made up of the set of all frequencies (conditional and unconditional) for the problem. Suffice it to say that the former space is, in general, many orders of magnitude larger than the latter. Thus, other things being equal, type-1 problems are easier to solve than type-2 problems.

The *practical* consequences of this are hard to determine. It seems to be the case that so-called 'real world' machine learning problems are almost never pure type-2. Thus, they can usually be solved by techniques that do not resort to exploring possible data reformulations (e.g. the ones tested in our study). Even learning problems that are intrinsically type-2 (i.e., which are constructed on

[7] Arguably, all problems that are pure type-2 are quasi-parity problems.

the basis of an input/output rule that implicitly invokes a reformulation step) may well exhibit 'spurious' type-1 regularity.

The example training set used above illustrates this. The problem is 'intrinsically type-2' since the input/output rule used to construct the pairs assumes the reformulation step of converting the original input variables to their difference. And yet the type-1 frequencies show some marked, non-chance values (see the frequencies for the cases <x2=1> and <x2=2>). These would be straightforwardly exploited by an algorithm such as Perceptron [14] or ID3 [4].

Even where intrinsically type-2 problems show very little spurious type-1 regularity they may still be solved by sophisticated learning algorithms such as backpropagation [6], cascade-correlation [8] or copycat [15].[8] It is, of course, well known that backpropagation can solve problems based on parity, symmetry or 'shift' relationships and all these typically involve the algorithm deriving what can be thought of as an internal recoding scheme.

However, we should not over-estimate the generality of such methods. All of them introduce restrictive assumptions about the nature of the type-2 regularity to be discovered. Backpropagation for example effectively assumes that the required reformulation can be expressed in terms of the user-defined architecture of semi-linear transfer functions, and that it can be discovered by the gradient descent method embodied in the learning algorithm. If the assumption is invalid, the learning necessarily fails.

This may help to explain why backpropagation usually *fails* to solve low-order parity problems when they are presented as generalization problems (i.e., when some cases are held back for testing purposes). The graph shown in Figure 7 was produced from an empirical survey that involved running backpropagation on 4-bit parity generalization problems (with four, randomly selected cases used as unseens) using a wide range of internal architectures, including the theoretical minimal architecture. All the curves in the upper half of the graph are error profiles[9] for the testing set of four cases. All the curves in the lower half of the graph are error profiles for the training set. There are 32 pairs of curves in all although many of them are bunched together in two clumps at the far left of the graph. Rather obviously, generalization over the testing cases was never observed to improve much beyond the chance level in any of the runs recorded. But the point to note is that the training-set error profiles typically go to zero rather rapidly. This tells us that the generalization failure occurs in the context of perfectly *successful* learning, i.e., perfect acquisition of the training cases. This is a particularly concrete sort of generalization failure since it cannot be

[8] This latter is not usually presented as a learning algorithm. However it can certainly be construed as such.

[9] The error measure is the average difference between actual and target activations. For these experiments we used standard learning parameters; i.e., a learning rate of 0.5 and a momentum of 0.9.

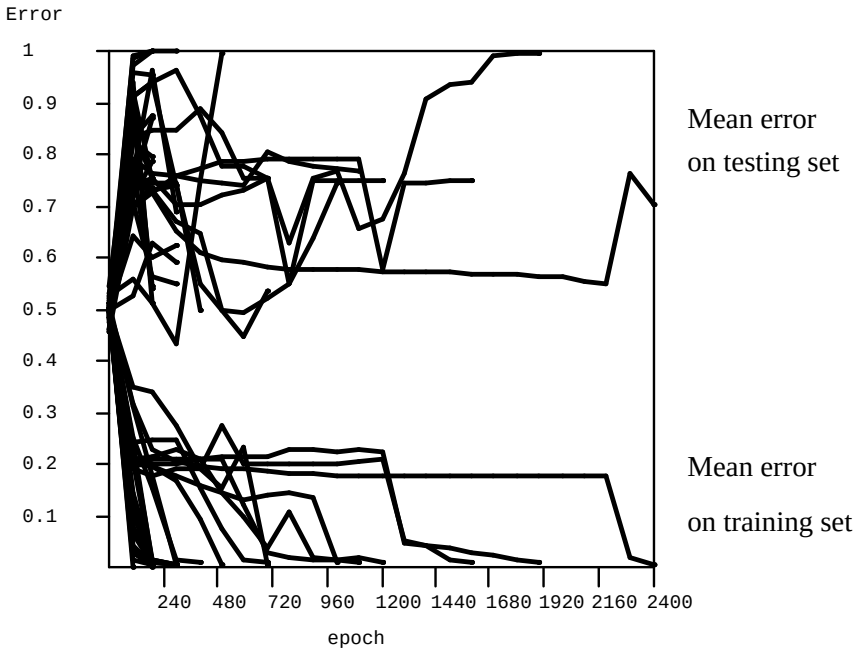

Figure 7: Generalization performance on 4-bit parity.

overcome by increasing the amount of training or by changing parameters. Once a supervised algorithm has learned the training cases perfectly, generalization grinds to a halt. As far as the algorithm 'knows', it is *already* producing perfect performance.

4 Measuring the degree of type-1 regularity

Given the overwhelming search complexity associated with the discovery of type-2 regularity, we can plausibly hypothesize that learning algorithms (at least those that rely on regularity exploitation) will be geared towards the exploitation of type-1 rather than type-2 regularity. Such algorithms can be expected to perform badly on problems that are primarily type-2. It is natural to wonder whether this might explain the results reported above. If it were the case that all the algorithms considered were in the class of 'type-1-oriented' learning algorithms, then the reason why they tend to perform badly on conditional-approach might simply be that this problem (as opposed to obstacle-avoidance

and pursuit) is actually based on a type-2 rather than a type-1 regularity. But do we have any good reason to believe this is the case?

4.1 Type-2 problems are based on relativistic rules

As it turns out, there is a way of measuring the degree of type-1 regularity in a given training set and we can apply this measure to show that simulation-derived training sets for conditional-approach *do* embody low levels of type-1 regularity when compared against training sets for the other two behaviors. However, before describing the measure let us first make some informal comments about the nature of the two regularity classes.

A (supervised) learning problem is always defined in terms of a target input/output mapping. In all reasonable problems, the mapping is based on an input/output 'rule' of some sort and it is the task of the learning to discover this rule and represent it in such a way as to enable unseen inputs to be mapped onto their correct outputs. In the simplest case, the rule refers (perhaps implicitly) to particular input-variable values. If it does so, we will expect to see correlations between particular input values and particular output values. In other words, the rule will tend to have a 'representation' in the form of strong, type-1, conditional-frequency effects.

But of course the rule may not refer — even implicitly — to particular input values. It may refer to *relationships* between input values. The parity rule is an obvious example. The parity rule does not 'care' about explicit input values. It only cares whether there is a particular relationship among them. And in this case, where the rule only takes account of input-value relationships, it will *not* have the effect of producing type-1 correlations in the training set. The rule has nothing to do with explicit values, so how could it?

On purely a priori grounds, then, we will expect to see training sets based on 'relativistic' input/output rules exhibiting low-levels of type-1 regularity. The obvious rider to this is that we will necessarily expect to see such training sets exhibiting *high* levels of type-2 regularity. If the input/output rule is based on a relativistic effect, then any recoding which effectively expresses the value of the relationship as a single variable value (as our differencing encoding did above) will automatically produce a strong type-1 effect, and thus a strong type-2 effect with respect to the original data.

This argument tends to favour the idea that the apparent hardness of the conditional-approach problem has something to do with it being based on a type-2 effect. Clearly, the input/output rule for conditional-approach *is* a relativistic one: it is based on the calculation of a *ratio* between two values, namely

the apparent width and the apparent closeness of the object.[10] 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 in both cases we are dealing with a non-relativistic rule; i.e., a rule that takes account of explicit values rather than relationships between them.

4.2 Measuring type-1 regularity

We find ourselves, then, increasingly in favour of the idea that the relative hardness of 'conditional-approach' is due to the fact that this problem is based on a type-2 rather than a type-1 effect. However, to be more certain of this conclusion we should attempt to *measure* the amount of type-1 regularity in the various training sets and show that conditional-approach does indeed differ in this respect. (Measuring type-2 regularity is out of the question since it would involve implicitly searching the whole of Turing-machine space.)

A measure of type-1 regularity must satisfy certain constraints. Obviously, it must be sensitive to the degree to which type-1 frequencies diverge from chance levels (since this is the essence of the effect in question) but it must not be insensitive to dependencies between these effects. In almost all cases, we will have many frequency effects associated with the *same* aspect of the regularity. Thus if we simply work out the overall divergence from chance-frequencies (e.g., as an average or total) we will compute a value which overstates the case.

To get over this problem we must measure divergence within a subset of independent frequency effects. One way to obtain such a set involves using Bayesian inference as an output-generation process. The procedure is as follows. First of all we compute all empirical frequencies. Then, we find the smallest subset of frequencies that — when treated as conditional probabilities — successfully generate (via Bayesian inference) correct outputs for all inputs. This operation necessitates a small shift in perspective. When we treat a frequency as a conditional probability we effectively make the transition from observing that case X is associated with case Y with frequency f, to stating that case Y exists when case X exists with a probability of f. Once we have made this shift we can use Bayesian inference straightforwardly to compute desired outputs from

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

given inputs.[11] We select a given input, we work out which cases it exhibits, and we then integrate the relevant conditional probabilities to derive probability distributions for the output values.

Finding the smallest subset of frequencies (probabilities) that completely 'captures' the training set (i.e., enables correct outputs to be generated for all inputs) is guaranteed to give us the set we want. Since the set is of a minimal size, we know that it must minimize (even if it does not abolish) the overall dependence between the effects. (If it did not, it would be possible to shuffle some cases in and some cases out to achieve a smaller set.) Since it captures the entire training set, we know that it cannot exclude any effect that reflects an aspect of the input/output rule.

Having derived a minimal subset of maximally independent frequency effects we still have the problem of measuring the overall divergence from chance-levels. It is not clear what tradeoff should apply between effects which diverge strongly from chance levels, and effects which diverge weakly but are nevertheless more general (i.e., useful) with respect to the capture of the training set. We can, however, finesse this problem entirely by observing that the relative *size* of the subset, relative to the original training set, will itself provide a satisfactory measure of the level of type-1 regularity. To compute the relative size we find the ratio between the number of variable instantiations used in the training set, and the number used in the frequency subset. With high levels of type-1 regularity we expect to see stronger and the more independent frequency effects. When such effects exist we will need fewer of them for a complete capture of the training set. Thus the relative size of the smallest subset that completely captures the training set measures the overall level of type-1 regularity.

A natural way to summarize this measure of type-1 regularity is as a compression ratio [16]. The compression ratio we define as the ratio between the number of variable instantiations used in specifying the frequency effects and the number used in the original training set. We then define the *empirical redundancy* of a particular training set to be the compression ratio that is achieved when we find the minimal subset of empirical frequency effects that completely captures the training set.

We can show how the measure works by applying it to the toy learning problem described above. This problem involves producing a 1 if the difference between the two input variables is 1. Our initial formulation of the problem contained just six cases and if we apply the measure to that training set we obtain a value that is strongly biased by the overhead costs of frequency-subset specification. However, we can derive a more realistic (i.e., larger) training set for the problem by allowing the input values to vary between 0 and 4. This gives us the training pairs shown on the left below.

[11] Of course, doing so entails assuming the statistical independence of variables.

Original pairs			Derived pairs				
0	0	--> 0	0	0	0	-->	0
0	1	--> 1	0	1	1	-->	1
0	2	--> 0	0	2	2	-->	0
0	3	--> 0	0	3	3	-->	0
0	4	--> 0	0	4	4	-->	0
1	0	--> 1	1	0	1	-->	1
1	1	--> 0	1	1	0	-->	0
1	2	--> 1	1	2	1	-->	1
1	3	--> 0	1	3	2	-->	0
1	4	--> 0	1	4	3	-->	0
2	0	--> 0	2	0	2	-->	0
2	1	--> 1	2	1	1	-->	1
2	2	--> 0	2	2	0	-->	0
2	3	--> 1	2	3	1	-->	1
2	4	--> 0	2	4	2	-->	0
3	0	--> 0	3	0	3	-->	0
3	1	--> 0	3	1	2	-->	0
3	2	--> 1	3	2	1	-->	1
3	3	--> 0	3	3	0	-->	0
3	4	--> 1	3	4	1	-->	1
4	0	--> 0	4	0	4	-->	0
4	1	--> 0	4	1	3	-->	0
4	2	--> 0	4	2	2	-->	0
4	3	--> 1	4	3	1	-->	1
4	4	--> 0	4	4	0	-->	0

The empirical redundancy of this training set turns out to be 14.6%. Reconfiguring the training pairs, adding in an extra input variable whose value is just the difference between the first two input variables, gives us a training set whose empirical redundancy is 89%, i.e., a great deal higher. The difference between these values clearly reflects what we already know to be the case: that the derived pairs effectively reify (in type-1 form) the type-2 regularity in the original pairs.[12]

When we apply our measure of type-1 regularity to simulation-derived training sets for the three learning problems we find — as we expected — that the value for conditional-approach stands out as a special case.[13] When we tested training

[12]The discrepancy between the two redundancies is particularly extreme due to the fact that in this case we added the extra variable rather than substituted it for the original two input variables.

[13]In computing these measures we adopted a probabilistic approach to case identity. In the conventional scenario, case identity is a clear-cut issue: any two cases either are or are

sets for pursuit, obstacle-avoidance and conditional-approach we found that the empirical redundancy of the latter was markedly lower, see Table 8. It is not quite as low a value as we see in the case of the toy example above. However, this is only to be expected since with a much larger training set and many more input variables we expect to see many more spurious type-1 regularities. The hypothesis regarding the type-2 nature of conditional-approach, then, seems to be borne out by this particular statistical analysis.

Behavior	Empirical redundancy
Obstacle-avoidance	86.6
Pursuit	93.1
Conditional-approach	63.3

Table 8:

5 How could conditional-approach be learned?

Given the fact that none of the learning algorithms tested seemed able to deal with conditional-approach, it is natural to ask what sort of algorithm is actually *required* for this problem. A trivial and not particularly informative answer states that what is needed is an algorithm that can explicitly or implicitly recode the original training data so as to 'convert' the relativistic effect underlying the training inputs into a non-relativistic effect. Trivially, this might mean recoding the original, seven-valued inputs so as to add an extra variable whose value is the ratio between the apparent closeness and apparent width of the object in the sensory field. To implement conditional-approach in this context the animat must freeze when the value of the extra input is above a certain threshold; i.e. it must behave with respect to this extra input exactly as the obstacle-avoidance animat behaved with respect to its main proximity inputs.

Since all the learning algorithms tested solved the obstacle-avoidance problem we have every reason to think that *all* would succeed on this recoded version

not identical. With the probabilistic approach, any two cases are identical with a given probability. This means that to derive the frequency value for a novel case (or the conditional frequency for two novel cases) we have to average over the products of specified frequencies and specified identity probabilities. This approach is computationally more expensive but it has the advantage of enabling training sets containing continuous values to be dealt with gracefully. By assuming that any data point identifies the peak of a Guassian probability distribution for the point's true location, we can derive the identity probability for any two cases by measuring the distance between them. This ensures that *similar* cases have similar frequency analyzes.

of the conditional-approach problem. Thus any one of our learning algorithms, provided with an appropriate recoding ability, should be able to solve the problem. In the case of classical (i.e. program-based) learning algorithms it is easy to imagine how the recoding might be achieved: it would simply involve an implementation of the steps for the relevant control procedure. In the case of a network-based learning the question of the implementation of the recoding is less obvious. The results with respect to the various network learning algorithms suggest that a single-net implementation of the behavior may be difficult to obtain. However, it turns out to be fairly straightforward to 'hand-code' a multi-network implementation of the behavior.

As we have noted, at the most basic level, conditional-approach is all about performing a size-discrimination given only range and location information. The size of an object in the sensory field is a relativistic property of the apparent closeness and the apparent width of the object. As objects get closer they appear to get wider. Thus the actual width of an object is a function of the ratio between the apparent width and the apparent closeness. An obvious decomposition of the size-discrimination task, then, involves computing the actual size of the object by finding the ratio between the apparent closeness and the apparent width.

We can build a subnet for that implements this decomposition as follows. The task of *calculating* the ratio can be roughly approximated by feeding activation values representing apparent closeness and apparent width into a unit with a bipolar activation function (e.g., *tanh*). If one connection is positively weighted and one negatively weighted the activation of the sigmoid unit will be 'flat' (i.e., zero) just in case the ratio between the two values is close to 1. By arranging for any activation, positive or negative at this unit to inhibit some other unit, which is biased on, we can effectively obtain a subnet that provides a feature detector for the case in which the ratio between the two values is at some particular level — the level being fixed by the ratio between the two original weights.

To make this subnet work as a 'largeness-detector' we need subnets that will compute the closeness and width values. In fact subnets for both of these subtasks can be obtained using conventional backpropagation training. Hand-coding is also a possibility. For example, given the assumption that objects in the sensory field have no holes, the apparent width of an object in the sensory field can be derived simply by counting up the number of non-zero closeness values.

Consider a subnet in which the activation of each input unit is just a proximity input (i.e., a ray-closeness value). Each of these input units sends activation to a single, unique hidden unit. The hidden unit biases are set so that they function as threshold units, turning off only if there is no input arriving from the input unit. Now, to obtain a subnet that computes apparent width we

need only establish positive connections from all these hidden units to a single, linear output unit. The activation of the output unit will effectively measure the number of activated hidden units and therefore of the size of the object in the sensory field.

This decomposition solves the problem of building a largeness detector. However, for a complete solution we need a network that integrates this detector into a more general structure that also serves to implement the various behavioral responses. A plausible approach might involve building an 'approach net' that could potentially serve the dual purpose of (1) implementing the basic approach operation and (2) carrying out the computation of apparent width. A possible architecture for this subnet is shown in Figure 8. The network has seven input units via which the seven proximity stimuli are received. Each of these inputs feeds directly on into a single hidden unit whose bias and threshold is set so as to achieve the zero-thresholding effect described above. Activation from these hidden units then flows on into the two, main output units. Each of these controls the drive applied to one of the wheels, with the amount of drive corresponding linearly to the amount of activation.

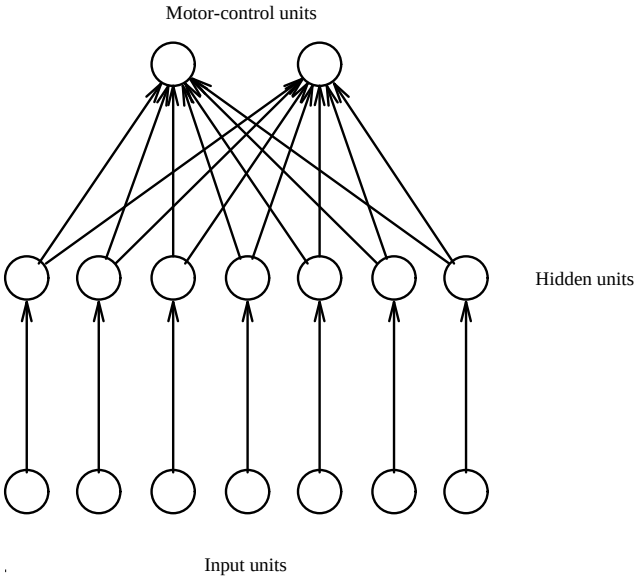

Figure 8:

The weights between the hidden units to the two main output units are arranged so that the further to the *right* (*left*) is the source of the input feeding a particular

input unit, the more it sends activation to the left-hand (right-hand) motor-control unit. The overall effect is that whenever an object appears in the sensory field the motors are activated in such a way as to make the animat turn towards the object. The further to the left (right) the object is, the more activation tends to activate the left-hand (right-hand) motor. (Weight configurations derived by learning may not necessarily follow this general schema, since there are other, less transparent ways of achieving the desired behavior.)

We can now add a set of identical positive connections to the approach net that carry activation from the seven hidden units to an additional output unit. The activation of this unit will measure the width of the object in the sensory field (as per the argument given above). We thus obtain the hybrid approach net envisaged; see Figure 9.

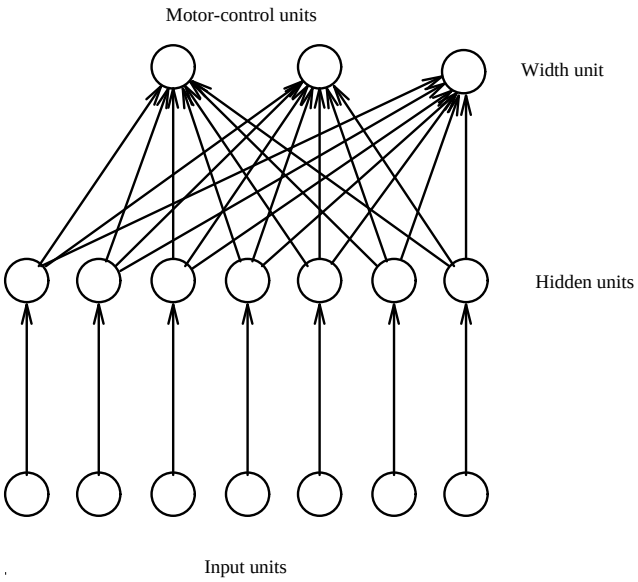

Figure 9:

To complete the implementation we can now add another three-layer subnet that computes overall closeness and establish connections from its (single) output unit and the width detector to a unit running a bipolar activation function (e.g. *tanh*). By feeding the output from that unit into a bipolar, zero-threshold unit (i.e. a unit that turns on only if it receives no input) we achieve a top-level unit that behaves as a detector for objects in the sensory field that exceed a certain size (as per our original decomposition). Obtaining the complete

implementation is now straightforward. We simply arrange for the size unit to (1) respond specifically to the largeness (as opposed to smallness) of objects and (2) to send inhibitory output to the motor control units. This provides us with a network that should implement the conditional-approach behavior quite successfully. The overall architecture is sketched out in Figure 10.

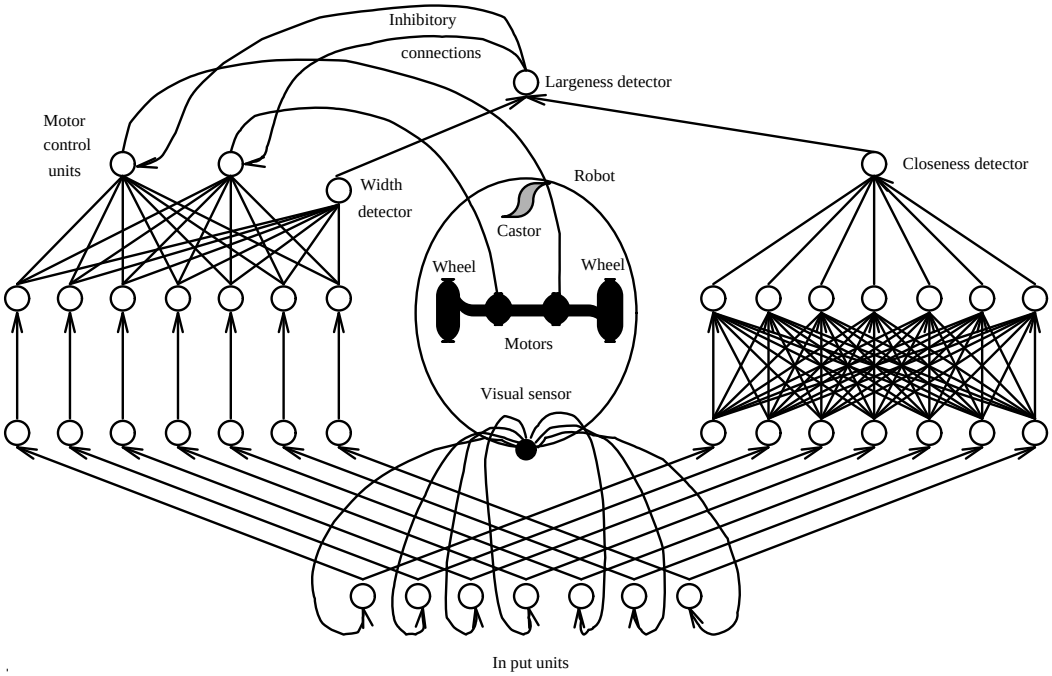

Figure 10:

Of course, speculating on ways in which learning algorithms might *implement* the required recoding for conditional-approach does not even address the fundamental issue, which is the question of how such recodings might be derived within the main learning process. At present, it seems to us that this task necessarily involves searching out some portion of the space of possible recodings (i.e., the space of possible Turing machines). But the complexity of this search is such as to suggest that any such algorithm would be doomed to failure. The fundamental question, then, remains very much open.

6 Summary and concluding comments

The paper sought to investigate the extent to which learning and evolutionary methods can be used to obtain simple, adaptive behaviors. It 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 learning problems may involve discovering some blend of two types of regularity and that problems that primarily involve discovering the more accessible (type-1) form of regularity are likely to be more easily solved in general. Formal and informal arguments were put forward to establish that the conditional-approach learning problem involves exploiting the less accessible, type-2 form of regularity. The implication was then drawn out that the learning algorithms tested were all primarily oriented towards the more accessible type-1 form.

References

[1] 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.

[2] Harvey, I., Husbands, P. and Cliff, D. (1993). Issues in evolutionary robotics. 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.

[3] Gibson, J. (1986). Artificial intelligence programming environments and the poplog system. In M. Yazdani (Ed.), *Artificial Intelligence: Principles and Applications* (pp. 35-47). London: Chapman Hall.

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

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] Rumelhart, D., Hinton, G. and Williams, R. (1986). Learning internal representations by error propagation. In D. Rumelhart, J. McClelland and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Vols I and II* (pp. 318-362). Cambridge, Mass.: MIT Press.

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

[15] Hofstadter, D. (1984). The copycat project. A.I. Memo 755, Masachusetts Institute of Technology.

[16] Held, G. (1987). *Data Compression: Techniques and Applications, Hardware and Software Considerations* (2nd edition). Chichester: Wiley.