Brave Mobots Use Representation

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QN
UK
Email: Chris.Thornton@cogs.susx.ac.uk
WWW: http://www.cogs.susx.ac.uk/users/christ/index.html
Tel: (44)1273 678856

December 20, 1995

Abstract

The paper uses ideas from Machine Learning, Artificial Intelligence and Genetic Algorithms to provide a model of the development of a 'fight-or-flight' response in a simulated agent. The modelled development process involves (simulated) processes of evolution, learning and representation development. The main value of the model is that it provides a clear illustration of how learning processes may lead to the formation of *representations*, and how these may form the infrastructure for closely-coupled agent/environment interaction.

1 Introduction

In recent years there has been a rapid development in our understanding of the detailed mechanisms underlying the emergence of intelligent behaviour. Research on Machine Learning (ML) has introduced powerful new models of lifetime learning (Shavlik and Dietterich, 1990). Research on Genetic Algorithms (GAs) and Artificial Life (ALife) has advanced our understanding of the role played by evolution (Cliff, Husbands and Harvey, 1993). And research in Artificial Intelligence (AI) and psychology has provided powerful illustrations of the way in which the emergence of intelligent behaviour depends on the development of representational structure (Muggleton, 1992; Karmiloff-Smith, 1992). Unfortunately, despite this progress, the 'big picture' is still elusive. Ideas from one community do not interface easily with ideas from another.

If we look at the relationship between learning theories and representation theories, for example, we find that the learning processes investigated by the Machine Learning community tend *not* to produce 'representational structure' of the type commonly studied by the AI community. In fact, very often, learning processes are not constructive in any sense. Where they do involve the generation of structure, it tends *not* to have any obvious representational interpretation. The interface between models of learning and models of evolution is equally problematic. Many of the more powerful models of learning tend to be supervised in nature. This means they assume the existence of an all-knowing 'teacher' — an assumption which is hard to accommodate within standard evolutionary scenarios.

This paper launches an all-out assault on this interfacing problem. It introduces a novel learning method ('explicitation') and shows how its unsupervised, constructive nature allows it to be used to link learning with both evolutionary and representation-development scenarios. The main body of the paper provides a practical demonstration of this in the form of a computational model of the development of a 'fight-or-flight' behaviour in a simulated agent (an 'animat'). It shows how evolution, learning, agent/environment interaction and representation-development effectively 'cooperate' so as to achieve a particular behavioural competence.

2 The fight-or-flight task

In the model presented by the paper, a simulated agent (an 'animat') develops the ability to produce a fight-or-flight response to a certain type of attack. The model involves evolutionary processes, learning processes and representation-development processes. And it shows how the developmental process is modulated by the changing structure of the agent/environment interaction.

The agent which forms the main focus of the model is known as the 'prey'. This agent has the sensory-motor properties of a simulated mobile robot or 'mobot' (cf. Nolfi, Floreano, Miglino and Mondada, 1994; Reynolds, 1994; Koza, 1992) similar to the widely used 'Khepera' mobot (see Figure 1) (K-Team, 1993; Jacobi, Husbands and Harvey, 1995). The prey has two wheels (not shown) situated at either end of a transverse axle, and a castor at the rear to provide a stable, tripod wheelbase. The wheels can be driven forwards or backwards at differing speeds thus achieving any blend of forwards, backwards and rotational movement.

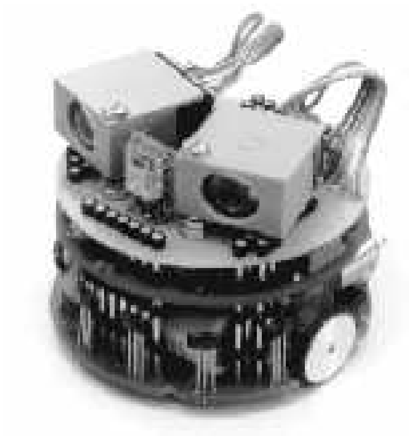
Figure 1: The Khepera mobot.

Within the model, the prey agent interacts with another, mobot-like agent, known as the 'predator'. The interactions take place in in a simple artificial world which is a flat, continuous, rectangular arena. The arena always contains one prey agent and one predator agent. In Figure 2 (a), these two agents are shown as circular shapes. The larger circle represents the predator. The smaller circle represents the prey. Each agent has an arrow which shows its current direction of motion.

The prey has a single range-finding sensor and a one-cycle memory (i.e., it is able to remember its

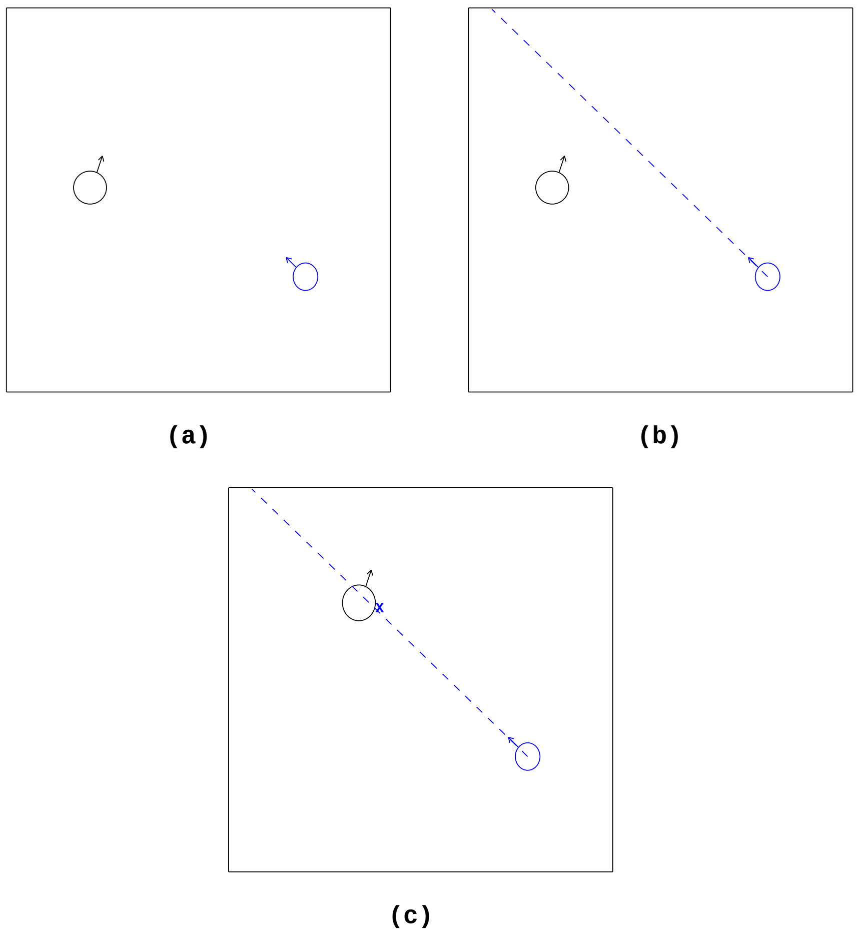

Figure 2: Basic agent scenarios.

previous sensor input). In each time cycle of the simulation, the sensor produces a real value between 0 and 1 which varies monotonically with the *proximity* of the nearest obstacle along a ray pointing directly ahead, see Figure 2 (b). The border of the arena does not constitute an obstacle and thus never affects the sensed proximity value. However, it may reflect the proximity of the other agent if the position of that agent falls somewhere along the sensor ray, see Figure 2 (c). If no obstacle intersects the sensor ray, the sensor returns a zero value. The larger of the two agents, known as the **predator**, has a variety of sensors. However, since the model is concerned with the development of behaviour in the prey, these are not relevant and will not be discussed.

Within the simulation, the predator's goal is to to destroy the prey as quickly as possible. The prey's goal is to survive while, at the same time, minimising movement. The predator attempts to achieve its goal by implementing a 'search-and-destroy' strategy. If it finds itself facing away from the prey, it turns towards it; see Figure 3 (a) and (b). (Note that in these displays, the lightly shaded agent shapes mark out previous positions, and thus constitute a visible trail.) If it finds itself facing directly towards the prey, it moves forward: Figure 3 (c). It may approach the prey at either a steady or accelerating pace. If the former, then it will swerve away at the last moment, provided the prey is facing directly

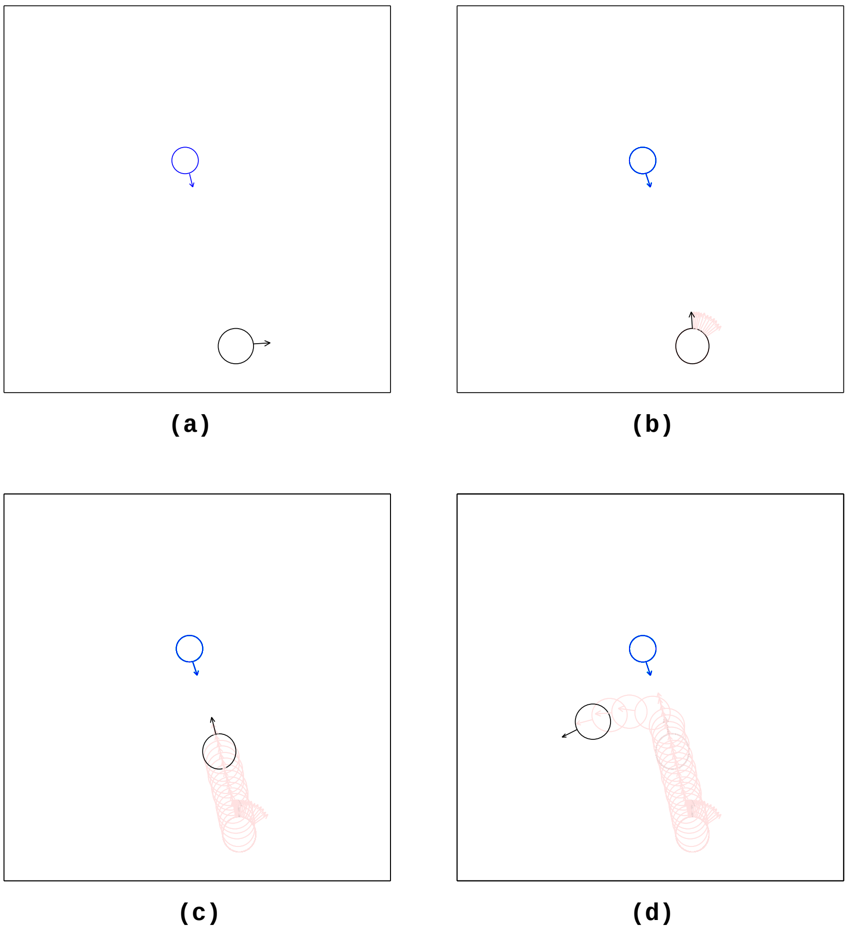

Figure 3: The basic predator/prey scenario.

into the attack, see Figure 3 (d). If it approaches at an accelerating pace, it will continue moving forwards regardless of where the prey is facing, eventually colliding with, and destroying it (not shown). Whenever a predator approaches, the prey should thus turn towards the approach and remain static. If the rate of approach turns out to be increasing, the prey should flee. The prey is thus required to implement a rudimentary 'fight-or-flight' strategy.

3 Previous work on fight-or-flight

What developmental or evolutionary story might account for the emergence of a fight-or-flight response in a particular agent? To date, work on fight-or-flight has largely been confined to the ethology community. This research has focussed on the *implementation* of the behaviour rather than its development (cf. Feder and Lauder, 1986; Gerlai, 1993). There is also work of relevance in the animal learning literature but this is concerned with the phenomenon of behavioural 'freezing' rather than fight-or-flight itself (cf. Bolles, 1979; Flaherty, 1985). Where the issue of the development of behavioural responses is confronted explicitly (cf. Barnett, 1973), there are few operational models and none as yet that deal explicitly with fight-or-flight.

Generic work on the evolution of intelligent behaviour has typically focussed on the genetic algorithm model of Holland (Holland, 1975; Goldberg, 1989) and the classifier system of Wilson (1991). Work on behaviour learning has concentrated on algorithmic models such as recursive decision-tree generation (Quinlan, 1986) or neural-network models such as backpropagation (Rumelhart, Hinton and Williams, 1986) or competitive learning (Rumelhart and Zipser, 1986). However, there has been no previous attempt to produce a computational model which shows the role played by evolutionary, learning and representation-construction processes in the development of this behaviour.

4 Evaluating standard learners

Can the development of the fight-or-flight behaviour in agents be modelled *solely* in terms of lifetime learning? To investigate this various learning methods were tested for their ability to acquire the fight-or-flight response. Experiments were conducted using C4.5[1] and standard backpropagation. Both of these are supervised methods and thus require explicit training examples. In the experiments carried out a training set of 1000 input/output pairs was used and these were derived direct from a running simulation.

The input in each pair was a vector of numbers corresponding to the prey's current sensory input and its corresponding input in the previous cycle. (The second input constitutes the learner's 'memory' of the first.) The output in each case specified the wheel motions for an appropriate execution of the fight-or-flight behaviour. Thus, in those cases where the input vector was derived in the context of an oncoming, accelerating predator, the target outputs specified the wheel motions for a flight response. In the testing of backpropagation standard parameters for learning rate and momentum (0.01 for learning rate and 0.9 for momentum) were used in an ordinary feed-forward architecture of three, fully interconnected layers. Architectures involving from 3 to 30 hidden units in the middle layer were tested, but it was found that the number of hidden units made little difference to the final performance.

The two methods were both tested for their ability to generalise. The quality of generalisation with respect to unseen cases was examined, as was the degree to which the learning methods were able to reproduce the fight-or-flight response in a replication of the original simulation. In both cases, both methods performed poorly. The average results obtained are summarised in Table 1. The error shown here is RMS error on a testing set of 1000 cases. The 'Deaths' column shows the number of deaths sustained by a trained agent in a simulation of 2000 time steps, in which the predator agent produced 12 aggressive (accelerating) approaches.

	Error	Deaths
C4.5	0.209	12
Backpropagation	0.670	8

Table 1: Performance of C4.5 and backpropagation on fight-or-flight.

[1] C4.5 is an improved version of the well-known ID3 algorithm (Quinlan, 1986).

5 A hand-crafted implementation

The failure of the standard learning methods to learn the fight-or-flight task might be due to the fact that this task is just 'too complex' to learn. But this seems unlikely since the behaviour is actually fairly easy to hand-craft. The essence of the fight-or-flight response is the discrimination between two *rates* of approach. For an agent with proximity sensors, an approach is essentially a pattern of increasing proximity values. An *accelerating* approach is a pattern in which the difference between the values is increasing through time. (Formally, an approach is a **first-order** dynamic effect (i.e., a change in a variable over time) while the accelerating approach is a **second-order** effect (i.e., a change *in a change* in a variable over time).) So any implementation for this behaviour must discriminate these two types of effect.

Let us consider how we might develop the implementation as a neural network. First, we must implement the agent's one-cycle memory and again the easiest way to do this is simply to arrange for the input stream to have a built-in 'echo'. We arrange for the input data to be a set of two-element vectors, such that each vector is made up of the current and previous sensor input. Given this input stream, our network must obviously have two input neurons. But how to arrange the rest of the network? One possibility is illustrated in Figure 4. In this network, the right wheel motor has a forwards bias and the left motor has a negative bias. There is also an internal neural network in which activation flows upwards (or 'backwards' through the mobot). The single proximity sensor is connected by an excitatory connection to the left input neuron of the network (the right input neuron contains the previous proximity input) and there are inhibitory connections from the main input neuron to both motors.

The internal structure of the network functions to provide the detection of accelerating approaches. The neurons in the first layer of the network determine whether the difference between the current proximity (labelled `p@t`) and previous proximity (`p@t-1`) is above or below some given value. The two neurons actually shown in the figure have one positive and one negative weight each. The absolute difference between the weights is 0.1 but the signs are different in the two cases. The left-hand neuron will thus be activated just in case the difference between the two proximity values is less than 0.1 (the label is therefore `d<0.1`) and the right-hand neuron will be activated just in case the difference between the two proximity values exceeds 0.1 (the label is thus `d>0.1`). When the difference is exactly 0.1, the two neurons are thus *both* inactive. Since they both have inhibitory connections to the neuron shown in the third layer, and since this neuron has a positive bias, it will always be activated except when the difference between the two proximity values is exactly 0.1. It thus forms a detector for this difference and its label is `d=0.1`.

This basic difference-measuring architecture is repeated (but not shown) for all other relevant difference values (0.2, 0.3, 0.4 etc.). Suitably weighted connections are then used to ensure that the activation of the left-hand neuron in the fourth layer precisely reflects the real difference between the two proximity values. The first four layers of the network thus function to measure the first-order dynamic effect we are concerned with; i.e., the current rate of approach as measured in terms of the difference between the current and previous proximity input.

Our ultimate aim is, of course, that the network should measure the *change* in the rate of approach, i.e., the acceleration. Fortunately, this can, be accomplished in almost exactly the same way we measured the change in the proximity. The top four layers of the network repeat the strategy implemented by the bottom four layers — only now using the measured difference value (i.e., rate of approach) as the basic input. The node containing the measured rate of approach is labelled `d@t` and the node containing the previous difference value is `d@t-1`. The top neuron of the network effectively measures the acceleration in the current rate of approach (`D`). The establishment of excitatory connections from this neuron to both wheel motors ensures that the mobot will 'jump' whenever an approach is made with a suitable

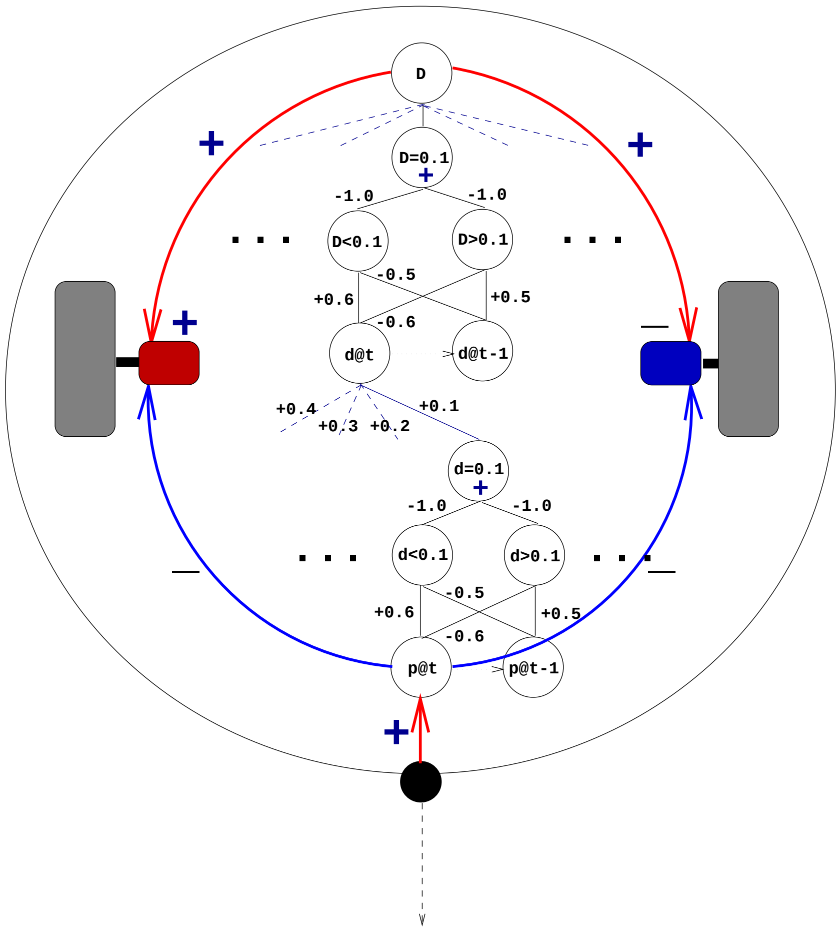

Figure 4: A hand-crafted fight-or-flight architecture.

high rate of acceleration see Figure 5. The network thus implements a rudimentary 'fight-or-flight' response.

6 Learning by explicitation

As we saw in section 4, the experiments with standard learning methods described in section ... produced poor generalisation performance. However, it turns out that we can obtain quite reasonable performance using a learning method called 'explicitation'. And by embedding this method within an ongoing evolutionary scenario, and by allowing processes of agent/environment interaction to take place within each agent's lifetime, we can actually obtain performance which is *reliably* good. The explicitation learning procedure is unusual in that it is both unsupervised *and* constructive. Moreover, the structures that it generates as a part of normal processing tend to have a representational interpretation. The method is introduced via a Bayesian analysis of induction.

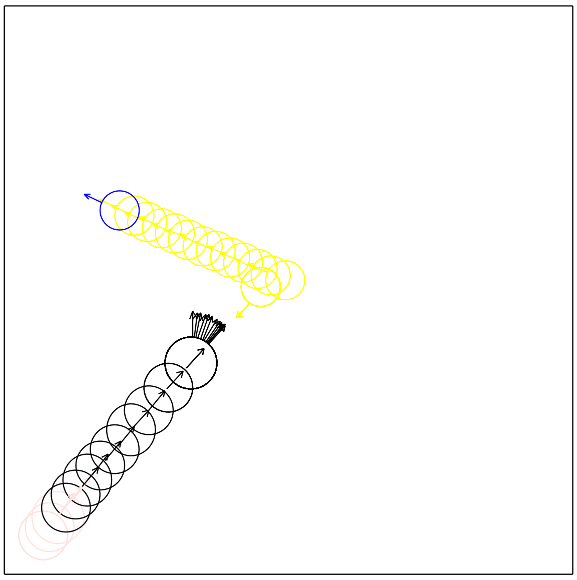

Figure 5: Flight response.

Imagine we have a body of data D. Each datum in D is made up of the values of variables $x_1, x_2, x_3 ... x_n$. Assume one datum has a missing value. Can we use the other data to predict the missing value? If we find that every possible value has an equal observed probability then we clearly cannot make any prediction at all. This occurs if, for all cases

$$P(x_i = v) = \frac{1}{|V|}$$

where x_i is the variable with a missing value, v is a particular value, and V is the set of all possible values of x_i. If all values do not have the same probability then we should obviously predict the missing value to be the one which has the highest observed probability. However, there are several ways in which we can work out observed probabilities. We can look at the unconditional probability of seeing a particular value of x_i.

$$P(x_i = v)$$

We can look at the probability of seeing a particular value conditional on explicit values of the other values, i.e.,

$$P(x_i = v_a | x_j = v_b ...)$$

where v_a and v_b are possible values and '...' denotes the optional inclusion of other instantiations.

Finally, we can look at the probability of seeing a particular value conditional on there being an *implicit* property among the instantiations of other variables:

$$P(x_i = v | g(X) = v_g)$$

Here X is the entire datum and v_g is the value of a function g, which evaluates the implicit property.

Methods which attempt to discover and exploit such probabilities for inductive purposes, without using any other source of information, are 'empirical learning' algorithms. There are a large number of these (Shavlik and Dietterich, 1990, Michalski, Carbonell and Mitchell, 1983, Michalski, Carbonell and Mitchell, 1986). However, the Bayesian analysis enables us to divide them up into two basic types.

A method that attempts to exploit either of the first two forms of probability confronts a relatively easy task. Only cases that are *explicitly* observed in the data need to be taken into account. There are a finite number of these. The task thus involves deriving frequency statistics (probabilities) over a *finite* dataset.

A method that attempts to exploit probabilities of the third form confronts a much harder task. It has to first identify the appropriate evaluation function for the implicit property (i.e., it has to guess what the property is). There are an *infinite* number of possible implicit properties and the task thus involves dealing with an infinitely large search space.

Practical learning methods naturally tend to be predisposed towards the easier task, i.e., they tend to exploit probabilities of the first and second form. A typical example is the Focussing method (Bundy, Silver and Plummer, 1985). Some methods such as ID3 (Quinlan, 1986) do not consider the third form at all. On the other hand, there are also methods which focus exclusively on the third form. Examples include the 'BACON-esque' methods of Langley and co-workers (Langley, 1977; Langley, 1978; Langley, Bradshaw and Simon, 1983; Langley, Simon, Bradshaw and Zytkow, 1987) and related methods such as (Wolff, 1978; Wolff, 1980; Lenat, 1982; Wnek and Michalski, 1992). Some methods such as backpropagation (Hinton, 1989) appear to straddle the fence, showing some ability to exploit both main forms (Thornton, 1994b).

Interestingly, we can deduce that the evaluation function used in the third form must measure a *relational* property of its inputs. To understand why, we need to think about the way in which the function differentiates different types of input. Let us say that the function produces a particular value whenever the input variables have certain *absolute* values. In this case, this evaluation is effectively a label for an explicit case. If all the values of the function are derived this way, the conditional probability can obviously be reduced to a set of probabilities of the second form. Thus, if the probability is a valid example of the third form, the evaluation function must measure a non-absolute (i.e., *relational*) property of its inputs. Learning problems whose solutions involve exploiting probabilities of the third form are thus **relational**. Problems which involve exploitation of probabilities for explicit cases are **statistical**, since they simply involve the derivation of frequency statistics over a finite dataset.[2]

Of course, since the space of relational effects is infinitely large, relational learners always and necessarily have a *bias* (Utgoff, 1986), i.e., they have a predisposition to consider certain types of relationship. Relational learners are also always potentially *recursive.* The identification of any set of relational effects involves the application of evaluations (functions) to the original data. This creates new values, and thus new data. These new data can themselves be processed for statistical and relational effects in a recursive manner.

At each stage, this process is effectively building a new level of description of the original data. Each level encodes or expresses a relational effect in the form of a single variable using an evaluation (a test

[2]Learning methods can be classified the same way. Learning procedures which exploit probabilities of the first and second form are statistical while ones which exploit probabilities of the third form are relational.

or measure) of the underlying relationship. Ths structure that will be built in a given case depends not only on the original data source but also on the bias of the method. Moreover, the influence of the bias 'accumulates' and becomes increasingly strong as the process builds layer upon layer.

I call any which exploits relational and statistical effects in this recursive manner an **explicitation process**, on the grounds that it incrementally renders implicit properties of the data *explicit*, through a process of recursive redescription.

6.1 A hybrid implementation

The explicitation process can be implemented in many different ways. However, for present purposes a 'hybrid' implementation (Torrance and Thornton, 1991) was used. This incorporates a neural-network component and an algorithmic or 'symbolic' component. The neural network component has the task of exploiting statistical effects and the algorithmic component has the task of implementing the recursive exploitation of relational effects. The latter uses a bias which effectively restricts its attention to relationships involving constant differences. A set of data were considered to exhibit a relational effect (i.e., to belong to a relationship) just in case they could be arranged into a linear order such that each variable would show a constant difference from datum to datum. Data satisfying this constraint were said to exhibit a **linear signature**. (As an efficiency measure, the relational exploitation was directed towards the statistical effects identified at any given layer, rather than to the relevant original data. Thus the learning always adopted a coarse-grained view of available data.)

The foundation for the neural network (statistical) component was the well-known unsupervised procedure of **competitive learning** (Rumelhart and Zipser, 1986). This is a general and robust method. However, like many unsupervised processes, it has a blind-spot in that it is unable to directly exploit low-order[3] statistical effects.[4] The basic procedure of an unsupervised learning method involves grouping inputs together according to similarity. This process 'discovers' cases in which a set of input values typically co-occur and thus tends to expose nth-order associations between variable values (where n is the number of values making up a complete input). However, the process *ignores* possible associations of order less than n, i.e., ones which do not involve the complete set of input variables taken together.

Unsupervised methods can usually be adapted to overcome this deficiency. In the case of competitive learning (Rumelhart and Zipser, 1986) the adaptation is straightforward; it merely involves 'double-weighting' each connection in the network (Thornton, 1994a). Each node in a competitive learning network defines a point in the input space known as a 'centre'. Each time an input is presented to the network, the action of the network serves to identify the centre which is nearest to the input, and then move it slightly closer still. Over time, the centres gravitate towards the central points for the densest clusters of inputs. Thus the operation of the method serves to identify clusters of similar inputs.

In the standard configuration, competitive learning works with some prespecified set of centres. Each of these is a vector V of n weights. The response of a particular centre to a particular input vector is just the negated sum of absolute differences between the input values and the weights:

$$-\sum_i |X_i - V_i|$$

For each input vector, the algorithm selects the centre with the highest response and then changes the ith weight by

$$r(X_i - V_i)$$

[3]The order of a statistical effect is the number of variables involved.

[4]An 'effect' for present purposes is simply any observable probability whose value deviates markedly from chance.

where r is the learning rate.

To sensitise the method to lower-order, probabilistic structure we associate each weight with an additional, confidence weight W_i. Each centre is thus defined in terms of a vector of weights and associated vector of confidence weights. We modify the response function so that the response of a centre to an input is

$$-\sum_i W_i(|X_i - V_i|)$$

and arrange for the confidence-weight vectors to be updated so that each W_i moves towards the normalised accuracy of the corresponding V_i. The unnormalised accuracy value for the ith weight of a particular centre is just

$$-\sum_j \sum_i |X_{i,j} - V_{i,j}|$$

where j ranges over the cases in the training set that are won by the relevant centre (node).

This 'double-weighting' adaptation provides us with the property we want. The original competitive learning process effectively searches for input clumps in the input space. The adapted method searches for clumps in *subspaces* of the input space. It not only finds associations between input values but also determines which values are associated. It is thus able to exploit mth-order statistical effects in an n-dimensional input space, where $m < n$.

6.2 Recursive exploitation of relational effects

To obtain the hybrid explicitation procedure, the adapted competitive learning regime was integrated with an algorithmic component capable of implementing relational searches. This component was configured so that it would become active at the point where all statistical effects in a given data source had been exploited. Its task was to test the discovered associations for the presence of linear signatures and, for each signature found, introduce a new variable whose values would provide an evaluation of the identified relationship. This was achieved by arranging for instantiations of the variable to reflect the projection of new data upon the relevant data line. These new variables, and their memory buffers, then form the input variables for the next layer of the network.

The method can be visualised as a network building operation which alternates between a statistical exploitation phase in which new competitive nodes are produced and a relational exploitation phase in which new variables are produced. This is visualised in Figure 6. Initially the network contains just two input nodes or variables. The double-weighted competitive learning process is then used to produce a layer of competitive nodes (Figure 6 (b)). (Competitive learning usually operates with a fixed set of nodes. However, for present purposes it was more convenient to allow nodes to be added incrementally until such time as all statistical effects had been exploited.) The relational exploitation process is then used to derive higher-level variables (Figure 6 (c)) which gives rise to the creation of a higher level of competitive nodes (Figure 6 (d)).

To summarise, the learning method used in the model is a variant of the competitive learning regime. It uses a recursive process of relational exploitation to incrementally extend the original neural network. The most important features of the method are as follows.

- Network connections are always double-weighted so as to enable exploitation of low-order statistical effects.

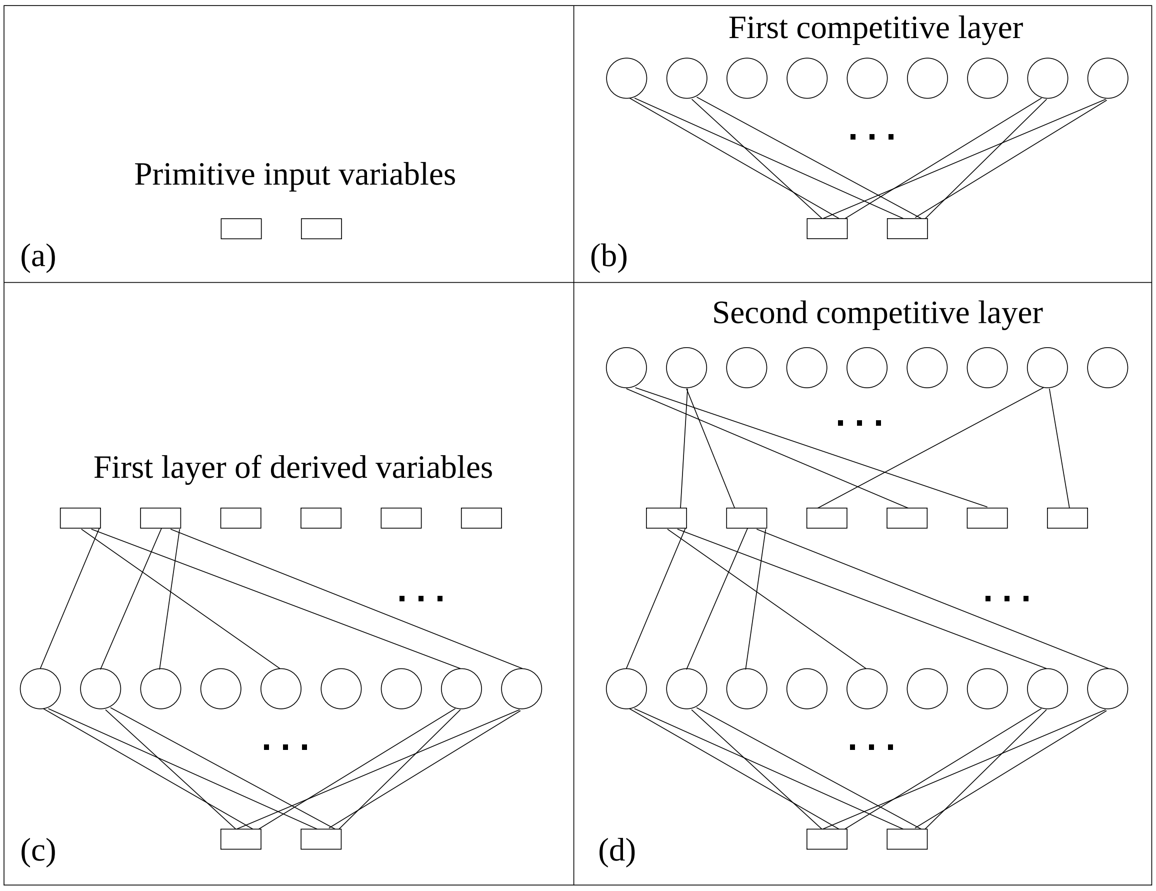

Figure 6: Explicitation using neural-network construction.

- The identification of a linear signatures at any layer always leads to the production of a new node at the next level of the network. The action of the network is configured so as to instantiate such variables with a value reflecting the geometric projection of the current input upon the line carved out by the relevant signature. Such variables thus provided an approximate measure of the relevant relationship.
- The learning is fully incremental. Nodes and layers are added to the network up until the point at which all effects — statistical and relational — are fully exploited.
- The learner's one-cycle memory is implemented by providing each main variable with a buffer which always hold its previous value.

7 The model

We now turn attention to the model itself. This takes the form of a simulation program written in the language POP-11 (Barrett, Ramsay and Sloman, 1985). The program takes about 10 minutes to run on a single-user Sun SPARC 1+. As we will see, the phenomena it generates while doing so include simulations of evolutionary processes, learning processes and interactions between predator and prey agents.

The simulation divides up into a number of phases. In the initial phase, the world contains a single prey agent and a single predator agent. The predator agent exhibits the 'non-aggressive' behaviour pattern described above. If it finds itself facing away from the prey, it simply attempts to turn towards it; see Figure 3 (a) and (b). If the predator finds itself facing directly towards the prey, it moves forward at a 'slow' or a 'fast' pace: Figure 3 (c). If the prey is directly facing the predator when the predator is about to collide with the prey, the predator turns away: Figure 3 (d). If the prey is facing away, the predator continues moving forwards, colliding with the prey and destroying it (not shown).

To begin with, the prey agent has a very simple internal architecture. It has a single proximity sensor and this is is connected by inhibitory or excitatory connections to either of the two wheel motors, both of which may have an independent bias. (Thus it is an example of Braitenberg's 'vehicle 2' (Braitenberg, 1984).) An example of this basic architecture is shown in Figure 7. This plan view shows the two wheels

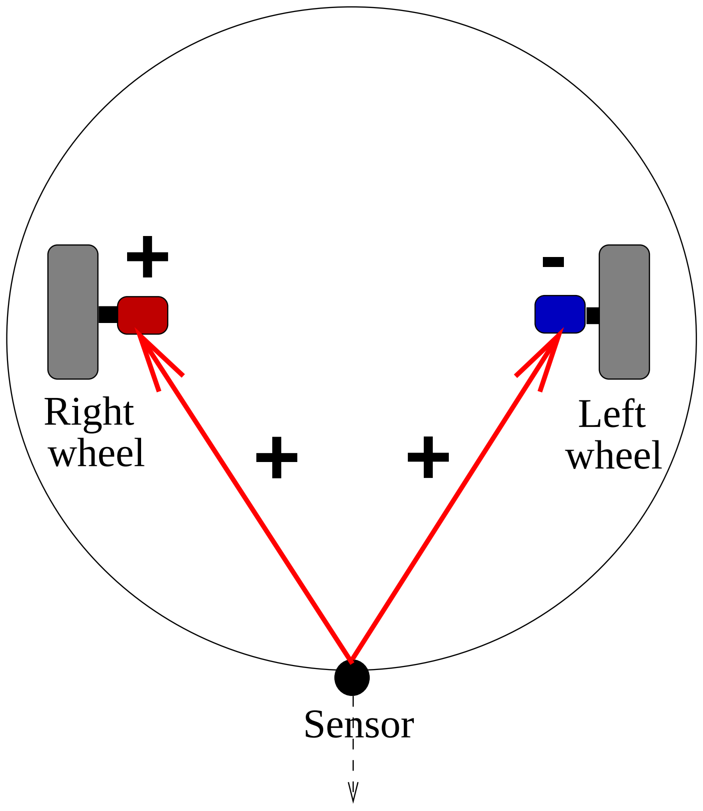

Figure 7: Early prey architecture.

and drive motors (the dark, rectangular shapes close to the boundary of the agent), the sensor (a small dark circular shape on the lower part of the boundary) and the connections between them. There is a plus sign above the right motor which denotes the fact that the motor has a forwards (positive) bias. Conversely, the left motor has a minus sign which denotes a backwards (negative) bias. There are excitatory connections from the sensor to both wheel motors. With these biases and sensor-motor connections, the agent will tend to move towards whatever excites its sensor, tending to veer to the left at all times (Braitenberg, 1984).

During the initial phase, the prey is subject to an evolutionary process involving asexual reproduction. The prey's internal architecture is specified using a simple genetic encoding, i.e., a 'genotype'. Each time a prey is destroyed through collision with the predator, a slightly mutated copy of its genotype is generated and this is used as the specification for an **offspring**. This offspring then replaces its parent in the arena. The predator agent remains unchanged throughout this process. We can think of this in terms of evolution having no effect on the predator. (The role of the prey's genotype in this initial phase is very limited: it only has to specify the existence/non-existence of the sensor-motor connections and the biases for the motors. However, the genotype length is variable and later in the model it is required to specify a more complex internal configuration.)

To survive attacks by the predator, the prey must rotate continuously until such time as the predator is sensed, and then cease all motion. Because of the forwards orientation of the prey's sensor, this strategy ensures that the prey stops at the point where it is facing directly towards the predator. This effectively 'blocks' any oncoming attack: Figure 8 (a). The only internal architecture that generates

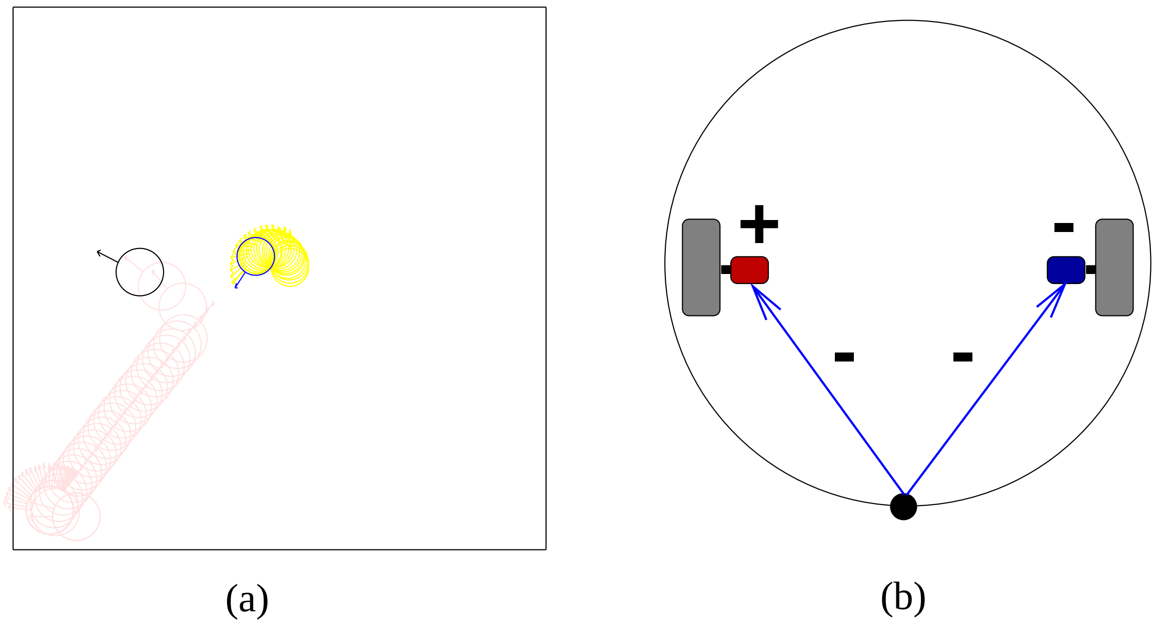

Figure 8: Simple survival strategy.

this behaviour is the one shown in Figure 8 (b). Here, the right-track motor has a permanent forwards bias (i.e., tends to run forwards unless explicitly stopped) and the sensor is connected by inhibitory connections to both track motors. Thus the prey is caused to rotate when there is no sensory input and to halt when there is. This implements the required survival strategy. The initial evolutionary phase continues until such time as a prey with this configuration of connections and biases is generated. From this moment on, the attacks of the predator are always successfully blocked. As a result the 'lifetime' of the prey is sufficient to enable learning to take place.

7.1 Events in the initial learning phase

As we have seen, the prey has a single sensor which senses the proximity of the nearest obstacle (e.g., the predator) along a ray pointing directly ahead. Predator attacks, in the initial phase of the model, take the form of rapid advances towards the prey, which are then aborted if the prey turns out to be facing towards the oncoming predator. Thus the prey's input environment is a sequence of proximity values, a sample of which is as follows.

```
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59
0.59 0.59 0.59 0.59 0.60 0.60 0.61 0.62 0.63 0.64 0.66 0.67 0.68
0.69 0.70 0.71 0.72 0.73 0.74 0.76 0.77 0.78 0.79 0.80 0.81 0.82
0.84 0.85 0.86 0.87 0.88 0.91 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

Note the prevalence of zero (0.00) values, the runs of increasing values (signifying an oncoming predator) and the runs of identical values (signifying a predator moving into a head-on position).

Learning takes place in the prey using the hybrid explicitation method. As mentioned, all data variables in the learning process are buffered, which means that the learning actually has access to a data stream based on two variables: the sensor value itself and the previous sensor value. The statistical-exploitation aspect of the learning leads to the discovery that particular pairs (i.e., two-value sequences) of sensor values occur with high frequency. Examples include (0.63, 0.64) and (0.77, 0.78). The existence of these is a consequence of the fact that the predator tends to advance on the prey at a steady rate, thus smoothly increasing its measured proximity from cycle to cycle.

The exploitation of these effects results in the formation of a number of competitive nodes (neurons). By the time the learning method has discovered all frequently occuring pairs of values, the network contains 33 nodes (see the lower part of the network shown in Figure 9). The learning process now begins to 'search' for relationships among the patterns (centers) that the neurons respond to. It quickly discovers that the patterns exhibit two linear signatures. The predator tends to approach at either a fast or a slow pace and these two styles of approach give rise to two signatures, one of which is associated with small constant case-to-case variable differences and the other of which is associated with large, constant, case-to-case variable differences.

The learning responds to the discovery of these two signatures by creating two **derived variables**. These are connected up to the relevant nodes come to form what are in effect virtual sensors — one measuring the proximity of a slowly-approaching predator and the other measuring the proximity of a rapidly approaching predator. Again, the learning introduces buffers for both derived variables, see the middle layer of the network in Figure 9.

7.2 The second learning phase

At this stage in the model, the predator begins to vary its behaviour. When approaching a prey, it now does so either 'aggressively' (i.e., at an *increasing* pace) or 'non-aggressively' (at a *steady* pace). The prey's sensory environment thus changes. A sample sequence of sensor inputs is as follows.

```
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59 0.59
```

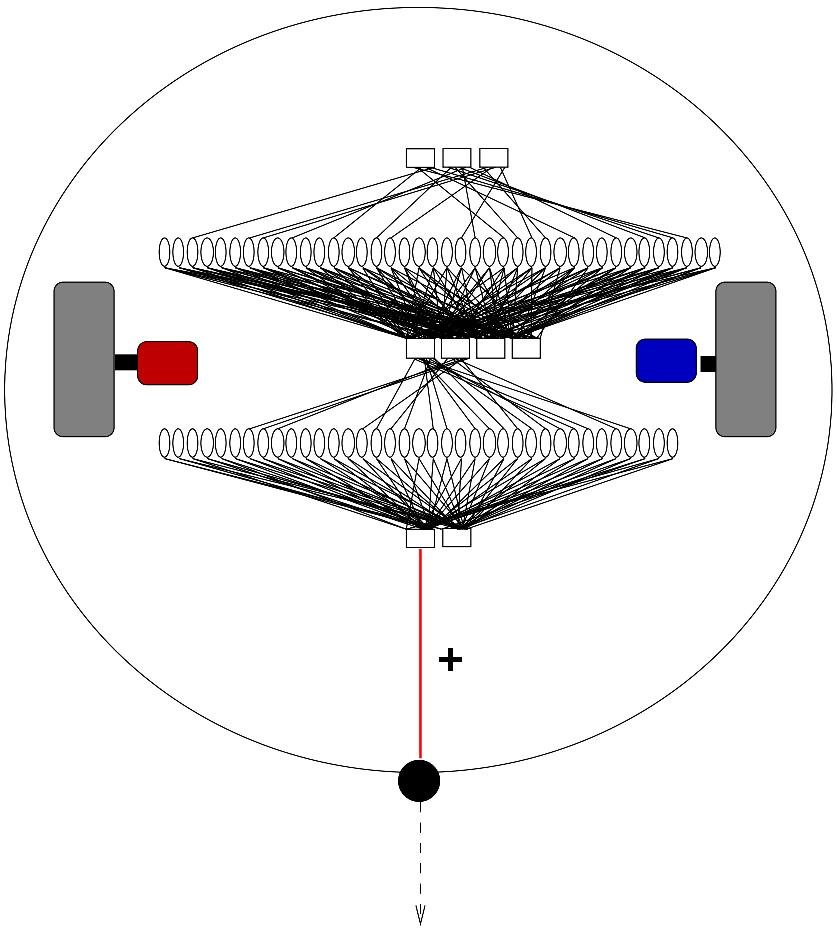

Figure 9: Final network structure.

```
0.59 0.59 0.59 0.59 0.62 0.66 0.72 0.73 0.76 0.81 0.83 0.87
0.93 0.95 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

In addition to the primitive input environment, there is now also the derived (recoded) input environment for the prey to consider. This is set of instantiations produced in the derived variables (and their buffers) which is produced when sensor inputs are fed to the network. When the learning is applied to these new data a further layer of nodes is created. Through training, these come to respond to specific patterns of values in the derived variables.

Recall that the first layer of derived variables (and their buffers) effectively measure the proximity of a particular type of approach (i.e., a fast or slow one). The main patterns that they exhibit (and which are therefore discovered by the learning) thus include the pattern created by an accelerating but distant predator, the pattern created by a constant-paced, distant predator, the pattern created by a close, accelerating predator and so on. Again, due to the fact that sensed proximity always increases, buffer values are always slightly below current values. Thus, the nodes responding to accelerating approaches exhibit patterns of values which show a marked relationship (another linear signature).

The detection of this relationship leads directly to the construction of a higher-level variable. This effectively measures the proximity of an accelerating approach and thus comprises an 'accelerating-approach'. Two other higher-level nodes are also generated at the same time. The final network is as shown in Figure 9 .

7.3 The final evolutionary phase

The generation of these final nodes exhausts the exploitation of statistical and relational effects. The final evolutionary phase now begins. Instantiations of the left, topmost variable reflect the proximity of an accelerating approacher. If no accelerating approacher is sensed, then this variable will have a zero instantiation. If one is sensed, this variable will have a non-zero instantiation. This node thus constitutes an accelerating-approach 'detector'.

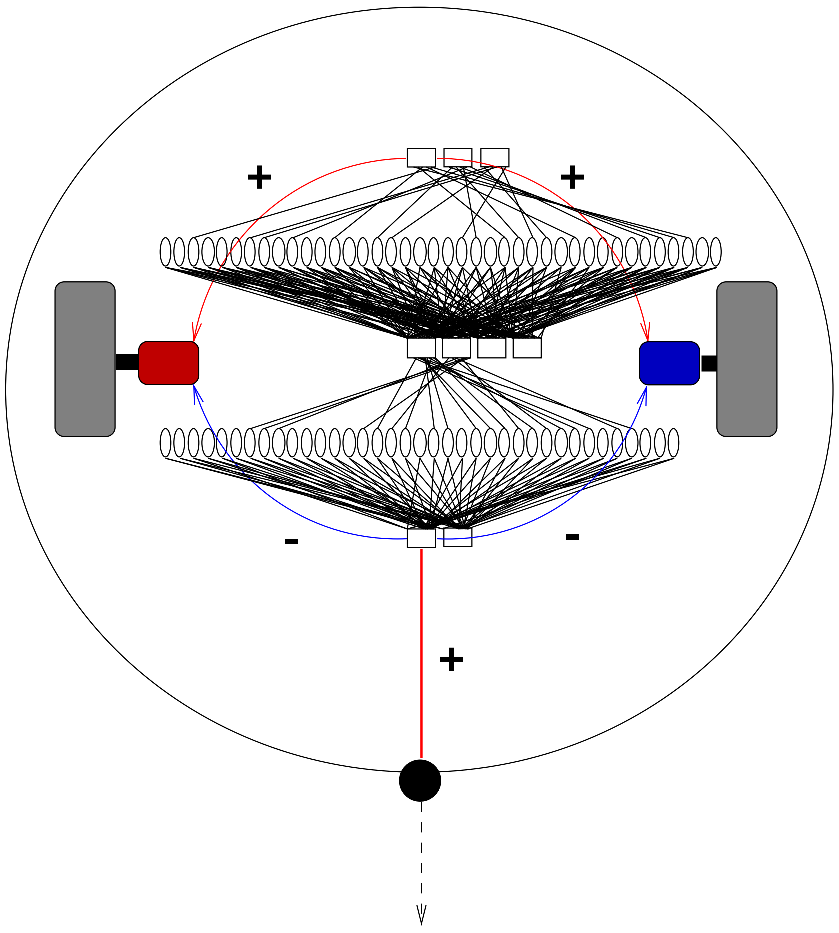

Figure 10: Final implementation.

The prey genotype is now considerably more complex. It may specify excitatory or inhibitory connections from any of the nodes in the network to either of the two motors. As evolution continues, different connection schemes are tried and many prey are destroyed. Eventually, a genotype is produced which specifies a combination of inhibitory connections from the main sensor variable, and excitatory connections from the accelerating-approach node to both of the two motors, as shown in Figure 10 . With these connections and biases, the prey 'jumps' whenever it is confronted with an obstacle moving at an accelerating pace directly towards it.

The final behaviour of the prey is illustrated in Figure 11. Its response to a steady-paced approach is to remain stationary (to 'face in' to the attack, see Figure 11 (a)) while its response to an accelerating approach is to jump away (Figure 11 (d)). Thus the prey implements a rudimentary of fight-or-flight response. With the successful emergence of this behaviour, the simulation terminates.

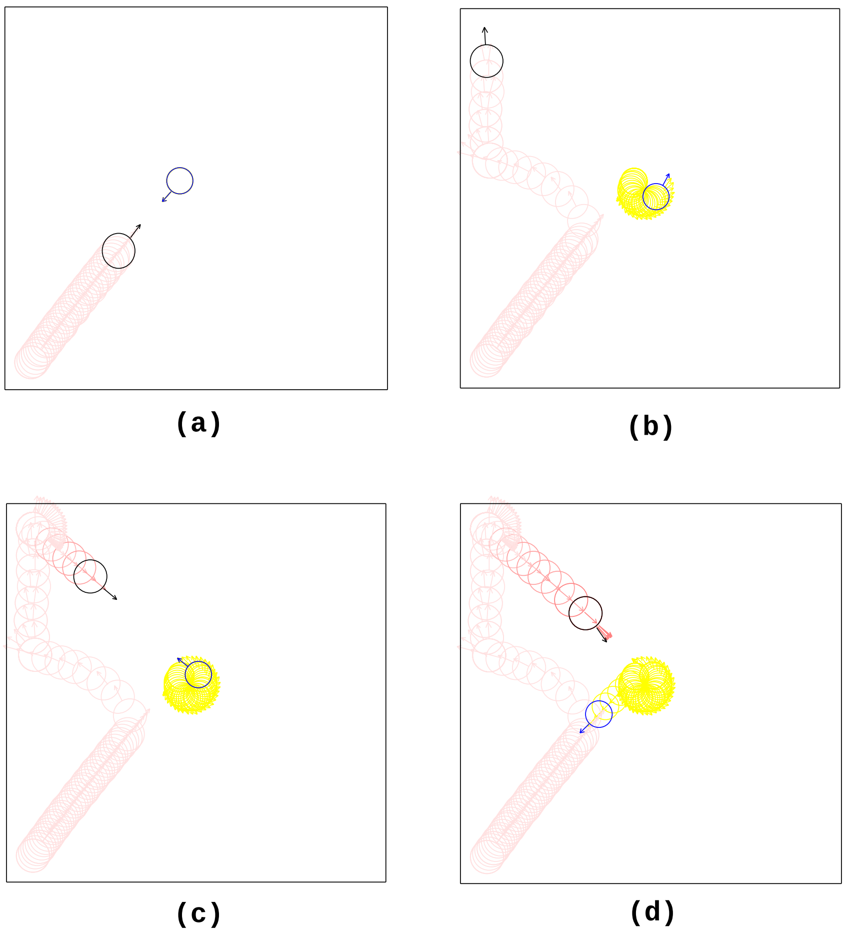

Figure 11: Production of flight response.

8 Representational interpretation

As we have seen, the learning and evolutionary processes simulated in the model result in the production of a large neural network structure within the prey. I now show that this structure can be given a representational interpretation. The internal structure of the network architecture developed in the prey agent(s) is illustrated schematically in Figure 12. Here, variables are shown as rectangular shapes

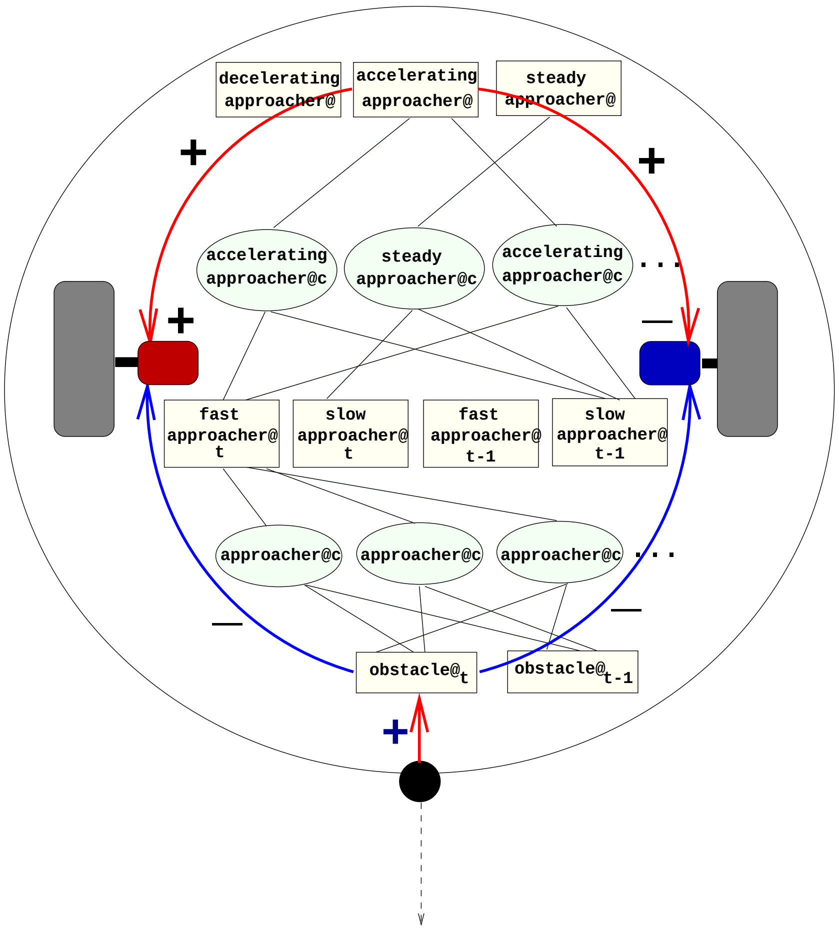

Figure 12: Representational structure of a successful prey architecture.

and nodes as oval shapes in the usual manner. All of the nodes and variables shown in the figure function as relational detectors or measures of some sort. All are given a label which attempts to characterise the property they measure or detect. The notations `t` and `t-1` are used to distinguish between current and previous properties. The character `c` is used to denote an unknown constant value.

The bottom, left variable is simply the channel for the current sensor input. Its label is thus `obstacle@t` since the sensor is an obstacle proximity sensor and thus provides a measure of the proximity of an obstacle at time t. Competitive nodes at the first level of the network respond to particular patterns of values across the main input variable and its buffer. Each one thus detects an approach taking place at a particular proximity; the label used is thus `approach@c`. (Note that all nodes at this level respond to events extending across time `t` and `t-1` and thus have no time subscript.)

The variables making up the third level of the network are derived from the signatures detected in the effects captured by the second-level nodes. They thus measure the proximity of an approach occurring at a particular rate (i.e., either fast or slow). The labels applied are thus `fast-approacher@t`, `slow-approacher@t` etc. The nodes at the fourth layer of the network respond to approaches which maintain a constant or a changing description over the third-layer variables, i.e., which either stay slow or fast, or change from one rate to the other. I therefore use the labels `accelerating-approacher@c` and `steady-approacher@c` here. Finally, the variables at the fifth and top layer of the network capture signatures among the effects identified at the layer below. They thus provide measures of the proximity of an accelerating, decelerating or steady approach. The labels used here are thus `accelerating-approacher@` `decelerating-approacher@` and `steady-approacher@`.

Is this 'representation' or not? If we address the question in a narrow-minded way, the answer has to be 'no'. The structure created by the simulated processes is not a 'representational structure' in the classical sense. It is not a frame structure. It is not an ISA hierarchy. It is not a script. It is not a Prolog database. And it is not a primal sketch, or an object-centered description.

On the other hand, the structure clearly *does* deal in terms of symbolic values. The current value of one of the `accelerating-approach@c` detectors, for example, is not an explicit (or even statistical) property of the raw input. It is a derived value which is used by other parts of the network — and the next layer of variables in particular — as a stand-in for an abstract property of the environment. The value is thus quite literally treated by the rest of the network as a symbol or sign of that property.

The same goes for the nodes and variables in layers two, three and five. Each node or variable in these layers effectively instantiates a symbol for an abstract feature of the environment. It does so by processing its inputs, which are themselves either raw sensor data or symbols for less abstract features of the environment. The network as a whole then is a kind of processing device. But the data it deals in are symbols for more-or-less abstract and in most cases *dynamic* features of the current environment. The dynamic state of the network is thus an active 're-presentation' — a multi-levelled recoding — of events and processes in the environment.

Rich and detailed though the final representational structure is, it nevertheless does not in any sense constitute a 'fully-detailed world model'. Nor is it imposed by any external agency. Rather, it emerges as a natural result of the interactions between processes and agencies operating both externally in the environment and internally within the prey. The main driving force behind the representational development in the prey is in fact the interplay between the assimilatory and accommodatory forces active within the lifetime learning (cf. Boden, 1979). Presented with sensor data, the network effectively *assimilates* that data to its current, internal representation. Over time there is also *accommodation*. This comes in two forms: structural accommodation, involving the creation of new nodes, variables and layers; and non-structural accommodation involving the tuning of network weights. As a result, there is the construction of a sequence of *redescriptions* or recodings of the prey's current sensory environment, much in the manner of Karmiloff-Smith's 'representational redescription' process (Karmiloff-Smith, 1979; Karmiloff-Smith, 1992; Clark and Karmiloff-Smith, 1993). The general conclusion then is that the structure *does* have a representational interpretation and that it is precisely the one shown in Figure 12 .

9 Final comments

The paper has presented a model of the development of a fight-or-flight behaviour in a simulated agent. Because the model makes use of the explicitation learning method, which is both unsupervised and constructive, it is able to present a developmental picture in which evolutionary processes effectively co-operate with learning processes in the formation of representational structure. Because the model uses processes which are fully incremental, it is able to show how an agent/environment interaction can modulate and guide an underlying evolutionary process. And because the learning is implemented in the form of a neural-network process, the learning process is at least reconcilable with current models of brain mechanism.

The model thus provides a 'big-picture' story about the way in which the development of complex intelligent behaviours might involve evolutionary processes, learning processes, agent/environment interaction and representation development. The learning method forms a a bridge between the GA paradigm on the one hand and the representationalist paradigm on the other.

Unfortunately, in its present manifestation the model has many shortcomings. As a computational implementation, it is not as robust as one would like. It also incorporates as ad hoc features the device of one-cycle memory (i.e., data buffering). The selection of a linear-signature bias during relational learning is also weakly motivated. But the most serious deficiency is probably the fact that the model makes use of what is in effect, a pre-scripted sequence of events. It requires that the predator implement certain changes in behaviour at certain points in the process. If these behavioural alterations do not take place on cue, the learning process is derailed.

However, it is believed that all these deficiencies will be remedied in the ongoing development of this work. Future versions of this model will aim to show how a fight-or-flight response emerges in a more natural, unscripted scenario, involving multiple predator and prey agents in addition to other forms of process and contingency. Work is currently in progress towards this end.

10 Acknowledgements

Thanks to Dave Cliff and Jim Stone for helpful input on an earlier draft.

References

[1] Barnett, S. (1973). *Ethology and Development.* Heinemann Medical.

[2] Barrett, R., Ramsay, A. and Sloman, A. (1985). *POP-11: A Practical Language for Artificial Intelligence Programming.* Chichester: Ellis Horwood.

[3] Boden, M. (1979). *Piaget.* Fontana Modern Masters, Fontana Press.

[4] Bolles, R. (1979). *Learning Theory* (2nd edition). New York: Holt.

[5] Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology.* London: The MIT Press.

[6] Bundy, A., Silver, B. and Plummer, D. (1985). An analytical comparison of some rule-learning programs. *Artificial Intelligence*, *27*, No. 2 (pp. 137-81).

[7] Clark, A. and Karmiloff-Smith, A. (1993). The cognizer's innards: a psychological and philosophical perspective on the development of thought. *Mind and Language, 8.*

[8] 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.

[9] Feder, M. and Lauder, G. (Eds.) (1986). *Predator-Prey Relationships. Perspectives and Approaches from the Study of Lower Vertebrates.* Chicago and London: University of Chicago Press.

[10] Flaherty, C. (1985). *Animal Learning and Cognition.* New York: Knopf.

[11] Gerlai, R. (1993). Can paradise fish (macropodus opercularis anabantidae) recognize a natural predator? an ethological analysis. *Ethology, 94* (pp. 127-136).

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

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

[14] Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

[15] Jacobi, N., Husbands, P. and Harvey, I. (1995). Noise and the reality gap: the use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J.J. Morelo and P. Chacon (Eds.), *Proceedings of the Third European Conference on Artificial Life* (Advances in Artificial Life). Springer.

[16] K-Team, (1993). *Khepera users manual.* Lausanne: EPFL.

[17] Karmiloff-Smith, A. (1979). Micro- and macro-developmental changes in languiage acquisition and other representational systems. *Cognitive Science, 3*, No. 2 (pp. 81-118).

[18] Karmiloff-Smith, A. (1992). *Beyond modularity: a developmental perspective on Cognitive Science.* Cambridge,Ma.: MIT Press/Bradford books.

[19] Koza, J. (1992). *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* Cambridge, Massachusetts: MIT Press.

[20] Langley, P. (1977). Rediscovering physics with bacon-3. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence: Vol I.*

[21] Langley, P. (1978). BACON.1: a general discovery system. *Proceedings of the Second National Conference of the Canadian Society for Computational Studies in Intelligence* (pp. 173-180). Toronto.

[22] Langley, P., Bradshaw, G. and Simon, H. (1983). Rediscovering chemistry with the BACON system. In R. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (pp. 307-329). Palo Alto: Tioga.

[23] Langley, P., Simon, H., Bradshaw, G. and Zytkow, J. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes.* Cambridge, Mass.: MIT Press.

[24] Lenat, D. (1982). AM: discovery in mathematics as heuristic search. In R. Davis and D.B. Lenat (Eds.), *Knowledge-Based Systems in Artificial Intelligence* (pp. 1-225). New York: McGraw-Hill.

[25] Michalski, R., Carbonell, J. and Mitchell, T. (Eds.) (1983). *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga.

[26] Michalski, R., Carbonell, J. and Mitchell, T. (Eds.) (1986). *Machine Learning: An Artificial Intelligence Approach: Vol II.* Los Altos: Morgan Kaufmann.

[27] Muggleton, S. (Ed.) (1992). *Inductive Logic Programming.* Academic Press.

[28] Nolfi, S., Floreano, D., Miglino, O. and Mondada, F. (1994). How to evolve autonomous robots: different approaches in evolutionary robotics. In R.A. Brooks and P. Maes (Eds.), *Proceedings of Artificial Life IV* (pp. 190-197).

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

[30] Reynolds, C. (1994). Evolution of corridor following behavior in a noisy world. In D. Cliff, P. Husbands, J. Meyer and S.M. Wilson (Eds.), *Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 402-410).

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

[32] Rumelhart, D. and Zipser, D. (1986). Feature discovery by competitive learning. In D. Rumelhart, J. McClelland and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Vol I* (pp. 151-193). Cambridge, Mass.: MIT Press.

[33] Shavlik, J. and Dietterich, T. (Eds.) (1990). *Readings in Machine Learning.* San Mateo, California: Morgan Kaufmann.

[34] Thornton, C. (1994a). Unsupervised learning with the soft-means algorithm. *Proceedings of the World Congress on Neural Networks.* Vol. IV (pp. 20-205). San Diego.

[35] Thornton, C. (1994b). Statistical biases in backpropagation learning. *Proceedings of the International Conference on Artificial Neural Networks* (pp. 709-712). Sorrento, Italy.

[36] Torrance, S. and Thornton, C. (Eds.) (1991). Special issue on hybrid models. *AISB Quarterly*, No. 78, AISB society.

[37] Utgoff, P. (1986). *Machine Learning of Inductive Bias.* Kluwer International Series in Engineering and Computer Science, Vol. 15, Kluwer Academic.

[38] Wilson, S. (1991). The animat path to AI. In J. Meyer and S.W. Wilson (Eds.), *Proceedings of the First International Conference on the Simulation of Adaptive Behaviour* (From Animals to Animats) (p. 16). Cambridge: MIT Press.

[39] Wnek, J. and Michalski, R. (1992). Hypothesis-driven constructive induction in AQ17-HCI: a method and experiments. ML1 Report, Center for Artificial Intelligence, George Mason University.

[40] Wolff, J. (1978). Grammar discovery as data compression. *Proceedings of the AISB/GI conference on Artificial Intelligence* (pp. 375-379). Hamburg.

[41] Wolff, J. (1980). Data compression, generalisation and overgeneralisation in an evolving theory of language development. *Proceedings of the AISB-80 conference on Artificial Intelligence.* Amsterdam.