Learning in Artificial Life: Conditioning, Concept Formation, and Sensorimotor Loops

Terry Stewart

Submitted for the degree of M. Phil.

University of Sussex

September 2000

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:

Learning in Artificial Life: Conditioning, Concept Formation, and Sensorimotor Loops

Terry Stewart

Summary

The first half of this thesis is a review of neural models of associative learning, with a particular focus on two things: the ability to form "concepts" (extracting patterns from the sensory data) and the capability of dealing with embodied agents. This full range of features is not readily apparent in any one model, but the architecture of these models is such that they may be able to complement one another in a combined system. The key models discussed are Edelman's Theory of Neuronal Group Selection, Hinton's Rectified Gaussian Belief Networks, and the University of Zurich's Distributed Adaptive Control.

The second half of this thesis details a closer examination of Distributed Adaptive Control, in light of the previous discussion. A series of experiments are performed on a re-implementation of this learning algorithm which compare its associative learning characteristics to those of the most basic of natural associative learning methods: classical conditioning. This includes the standard aspects of acquisition, extinction, generalization, and specialization, as well as studying the sorts of concepts (i.e. regularities) it is able to develop.

The result is that while Distributed Adaptive Control may show the surface capabilities of classical conditioning (the ability to have a conditioned stimulus act as a predictor for an unconditioned stimulus), it does not have the deeper abilities of classical conditioning. In particular, the types of concepts it can extract are limited by having to associate large values of one sensor type with large values in another sensor. Also, it does not exhibit the property of extinction, meaning that it cannot lose associations which are no longer valid.

It is hoped that this thesis provides the research community with two things. Firstly, it presents an experimental regimen for evaluating various aspects of classical conditioning, and shows how one particular algorithm fares under these tests. Secondly, the results of these tests highlight those areas which future work needs to examine, if we are to eventually arrive at a model with the important properties of natural associative learning.

Submitted for the degree of M. Phil.

University of Sussex

September 2000

Contents

List of Figures

Chapter 1

Motivation

My personal goal in working in this field is to understand life. Admittedly, this is a rather far-fetched goal, but this does not stop me from using it to shape the path that my research follows. Intertwined and inseparable from this goal is the creation of life (or, possibly equivalently, creating simulations of life). I believe that a significant part of understanding something is in the ability to actually make it. I believe that we cannot really understand what life is until it is possible for us to create it, and that along the way towards this goal our conceptions of the fundamental characteristics of life will change drastically. In particular, I feel that developing models of the style of learning observed in animals and then in people can lead to important and deeply useful insights into who we are and what we do.

1.1 Artificial Intelligence and Artificial Life

"Life" and "Intelligence" are highly linked terms. Furthermore, they resist any sort of rigid definition. However, we do generally agree that living things seem to exhibit various different characteristics that we call intelligent. "Intelligence", however, seems to be something that can have different levels: we feel reasonably comfortable saying that some things are more intelligent than other things, but we seldom say that one thing is more alive than another. One must, of course, note that we are arguing here from introspection, but it still seems to be an interesting distinction.

In the field of Cognitive Science, the term "Artificial Intelligence" tends to mean high-level intelligence. This is the sort of intelligence associated with human beings, and generally involves symbol manipulation (i.e. some sort of "internal language of thought" that the creature uses to reason and solve problems). More recently, this has been referred to as "Good Old-Fashioned AI", or GOFAI. This research has led to remarkable advances in things like chess-playing software, and has shed insight into how real creatures use (and generally do not use) logic in their problem solving.

“Artificial Life”, on the other hand, is associated with much more low-level behaviour. It typically studies creatures such as insects and takes the approach of combining a number of different very simple rules of action together in order to produce more complex and “life-like” behaviour. One of the greatest insights out of this approach is the observation that extremely simple rules can produce very unexpected and complicated behaviour. The planning and reasoning approach taken by the GOFAI researchers seems to be completely unnecessary for a great many tasks, such as avoiding obstacles, finding optimal routes, or catching a thrown ball.

Whether Artificial Intelligence includes Artificial Life, or vice-versa, is a debate that I do not wish to discuss here, as it is well beyond the scope of this paper. Some people want to expand the definition of Artificial Intelligence to include Artificial Life, and other people feel that it is obvious that the study of creating life includes the study of creating intelligence. I believe the question itself is meaningless, as it is an attempt to impose a rigid hierarchy on two concepts that do not have any widely accepted definitions. At the current point in my research, I am drawing from work that has generally been called Artificial Life (and I expect to continue to work at this level for some time into the future). I take Mother Nature as a guide here: very simple life forms existed long before creatures capable of complex symbolic planning behaviour. This observation leads me to believe that the behaviours exhibited by more complex creatures are built upon the behaviours exhibited by simpler creatures. It therefore makes sense to study simplistic behaviours first, before moving up to higher levels.

Chapter 2

Learning

As revealed by the title of this paper, I am primarily interested in the study of learning, as it seems to be fundamental to both life and intelligence. However, before we talk about learning in artificial life situations, we must first take a closer look at what is meant by the term “learning” itself.

2.1 What is learning?

To start our discussion of the term “learning”, let us take a highly general definition of the word, and then narrow our focus. “Learning” clearly refers to some sort of change within an organism. Furthermore, that change within the organism should change the behaviour of the organism (otherwise, the effect of the learning is not measurable). However, we are not interested in directly physical changes, such as the change in a bird’s flying ability when its wings are clipped. This leaves us with internally-caused changes: changes which are indirectly caused by a creature’s stimuli.

I do wish to point out that this definition quickly leads us into a conflict between the terms “learning” and “development”. Certain internal changes within an organism seem to happen regardless of the stimulus. This includes not only such things as bone growth, but also the development of the nervous system. An incredible amount of development occurs during the gestation period of most living creatures, and this occurs with very minimal (if any) stimuli. However, when we consider such things as the development of the speech centres of the human brain, we quickly discover that this development does not occur without the stimulus of human speech.

Even though development may require certain basic sensory stimulation, it does not seem to be driven by that input – rather, it makes use of the sensory data if it is available. This leads us to conclude that we should use “development” to refer to aspects of internal change that are driven by genetics, and “learning” to refer to aspects of internal change driven by individual stimuli. It is very important to note that the stimuli that are required by “development” must be evolutionarily stable. That is, they should be present in the normal environment of the creature. This allows us to say that we develop our ability to speak (since the presence of human speech something that is constant on an evolutionary time-scale), but that we learn that “dog” refers to a four-legged barking mammal.

For the purposes of this thesis, we are more interested in the individual “learning” aspect rather than the “development” aspect, but it may not be possible to completely separate the two. After all, how can we expect to talk about learning what “dog” refers to without mentioning the general fact of the gradual development of the human speech centre?

2.2 Why is learning necessary?

At first glance, it may seem odd that it is necessary for creatures to learn. Why is it that living things slowly develop skills over their lifetime? Wouldn’t it be much better, from an evolutionary point of view, for newborn infants to already have the skills to walk? (Note that this is, in fact, the case with some animals). Why not have full adult functionality straight from birth? Surely this would be an evolutionary advantage.

The simple answer to this question is that we actually don’t want behaviours to be prespecified. Living creatures exist in a highly complex and changing environment. Behaviours that are suitable in certain environs are not fit at all in other situations. Humans living in cold climates should certainly learn different patterns of behaviour (like putting on something warm before going outside) than humans living in warm climates. Cats living in a house should learn different patterns (like not scratching the furniture) than cats living in the wild. If behaviour was predetermined by genetic codes, these sorts of adaptive abilities would not exist. In fact, the ability to learn during an individual’s lifetime would seem to be a survival advantage to that individual, so it is unsurprising that the process of evolution has led to adaptable behaviour. Of course, we should remember that there may also be drawbacks to taking the learning approach, such as having to maintain a more complex nervous system.

2.3 What do things learn?

Up to this point, we have been talking about learning just as “change”. This is a fairly nebulous term, and it is interesting to examine exactly what this change is.

2.3.1 Optimization

The simplest form of change is a sort of “optimization of parameters”. In this fairly simple form of change, there is a known range of options available to the creature, and whatever learning algorithm exists simply has to choose from the available options. The standard example of this optimization process would be the basic genetic algorithm approach, where a large number of different possible values are chosen (i.e. a large population is created with randomly chosen parameters), and those that are successful survive to form the basis of the next generation. In this case, the optimization is happening over the lifetimes of many individuals, and the "best" (or "sufficiently good") parameter settings are those that allow the creature to survive and breed.

We can also do this sort of optimization within the lifetime of a single individual. The organism can by itself try out different options and learn from experience what the best option is. This process does require the use of a slightly more limited range of options, as if the penalty for trying out a "bad" option is fatal, then the creature cannot learn from its mistakes. One reasonably well-studied example of this process would be the adaptive orienting of a barn owl.

The owl uses both auditory and visual cues to find its prey. Normally, the owl first hears a noise, then orients its eyes in that direction. The difficulty is determining what that direction is, based on the sounds received by the owl's two ears. To complicate matters, the translation from sounds to direction is dependent on the geometry of the owl's ears and the size of the owl's head, both of which change during the lifetime of the owl. The internal process of connecting these two sensory modalities must adapt to the changing situation. However, this adaptation is within a fairly narrow range of possibilities, as it is not going to start using a completely new system for orienting. Thus, we can view this form of learning as a form of complex optimization. For details of a computer model based on this owl behaviour, see [28].

In this form of change, the creature need only find which option (or range of options) is "best". Defining what is "best" is generally seen as the role of the organism's genetics. For the owl, having the eyes and ears able to focus at the same point is considered best, and so the learning algorithm is based on that property. In this case, it is considered "good" if the owl, after hearing a sound, looks in a direction which then exhibits movement.

This approach has been very successful, as demonstrated by the popularity of genetic algorithms. The optimizing behaviour of genetic algorithms (or, more correctly, the "robust satisficing" behaviour, as they tend to find a range of "good enough" answers, rather than the one best one) gives a strong argument for their importance in living organisms. However, this approach to learning does not seem to capture the range and diversity of the sorts of learning generally associated with living creatures, as most genetic algorithm approaches are limited to learning within a specific range of possibilities. It could perhaps be argued that more complex learning is simply optimization of extremely complicated, interrelated parameters, but that does not seem to lead us towards a better understanding of the process.

This optimization approach can also be used to describe much of the research into neural networks. As will be discussed in more detail later, most neural network models keep the number of neurons constant and learn the best connection weights between them. However, some models allow the use of new neurons in a manner which starts to walk the line between development and learning (see section 3.3).

For an opposing point of view, the reader is directed to Dorigo and Colombetti's work in Robot Shaping [8], which uses a genetic approach to generating production system rules to allow a robot to learn behaviours in a manner based on operant conditioning in living creatures. This work is not examined in more detail in this paper, since I wish to focus on neural and physiologically plausible models of learning.

2.3.2 Concepts

A more intuitive way of thinking about what things learn is by looking at concepts. This tends to be what we think of when we talk of someone learning something. This applies to both high-level abstract learning of facts (for example, to learn about computers, we learn the concepts of a screen and a keyboard, input, output, data storage, and so on) and low-level learning of associations (the concept of "fire" being "painful" and thus "bad", or an animal often finding food in a particular location). Both the concepts and the relationships between the concepts are learned.

However, there are some rather unexplained issues with this approach. Where do concepts come from? How do we form new ones? How can we make them both stable and flexible enough to be useful? What types of relationships between concepts are there? How do we use these in such a way as to make decisions about what to do? These questions are generally referred to as the grounding problem, and point out that something more is needed to make a complete theory. However, the idea of a "concept" is so pervasive and useful that it seems to be a vital part of any theory of complex cognition and learning.

Of course, certain aspects of cognition do not seem to be suitable for being looked at as "concepts". The emotional experience of hearing a piece of music, for example, would seem to be rather difficult for one to describe in terms of concepts. However, at the moment we will not deal with this somewhat esoteric point, in the hopes that dealing with this problem is not fundamental to our current level of exploration of cognition.

In this paper, we will look at learning methods which work by extracting what can be thought of as concepts (or patterns) from the environment. All such methods will not be covered: instead, we will look only at the neurally-based models. For an overview of these models we will not be examining, see [10], which looks at these models from a GOFAI perspective.

2.3.3 Facts

Once we have concepts, we can also learn a very particular sort of relationship among concepts: facts. "The pyramids are in Egypt." "The earth goes around the sun." These sorts of statements can form the basis for highly complex, organized information processing and, arguably, represent the current peak of complexity and utility for life forms on this planet. As such, they are completely outside of the scope of this thesis. This sort of learning seems to only happen within humans and possibly within a very few other mammals. There is no consensus as to how this sort of learning occurs, or even on how we should think about these syntactic statements. For this thesis, I follow the idea that perhaps once we can understand the more basic forms of learning, that will shed some insight into how we can build up to this level of complexity.

2.3.4 Associations

However, there is one particular way of combining concepts which is of particular relevance for this thesis. Certain concepts can be *associated* with each other. As mentioned in the section on concepts (see section 2.3.2), we can learn to associate fire with pain. Of course, this simple example is something that could conceivably be specified genetically, since it is an association that should be learned by all creatures within a species that would encounter fire.

A more complex example could be learning about a new mildly poisonous plant by associating eating it with physical discomfort. Most of the original research in experimental psychology was in investigating these sorts of associations. In the section on classical conditioning (see section 3.2), we will discuss this type of learning in more detail.

We have thus arrived at a clearer picture of the sort of learning that is under consideration for this thesis. We are looking for a system which allows for the formation of associations between learned concepts, based on the creature's interaction with its environment.

2.4 How can we model learning?

However we end up representing the things we learn, there is the (perhaps) more important question of how this learning takes place (actually, these are two parts of the same question). Other than in formal education situations, we are not directly presented with facts to learn; we must infer them somehow from our surroundings. How this can be done has long been the question analysed by developmental psychology, in both animals and humans.

2.4.1 Traditional Approaches

The constructivism approach (as exemplified by Piaget) maintains that creatures start with a certain base set of skills and abilities, and slowly build onto this set in various ways. We learn new topics by relating them to old topics. Thus, the genetic code needs only to specify an initial "core" system, and the developmental processes control the rest of the creature's abilities. There is also an emphasis on developmental "stages", sudden jumps in a creature's ability. Constructivism regards this as what happens when a new set of abilities is formed.

Nativism, on the other hand, argues that the basic structures of the brain are much more innate. Learning is not about building onto the existing structure, but about fine-tuning it (much closer to optimization). Fodor, for example, portrays a view of nativism where the brain is divided into basic modules right from the beginning, and the modules are rigidly fixed into certain patterns of operation, and specific means of communicating between one another. As well, nativism tends towards domain-specific behaviour, and arguments for it tend to use examples from the animal world. For example, observing that rats more readily associate being sick with food than anything else makes it likely that they have some sort of innate (native) structure that makes such associations more readily.

For a more comprehensive overview of these theories from the perspective of modern cognitive science, the reader is directed to [29]. Work on merging the two theories can be found in [18]. The Rutkowska paper in particular uses the same sorts of paradigms and world-views as will be found in the rest of this paper.

2.4.2 Dynamic Systems

More recently, a few researchers have been looking for a completely different way of looking at development. The hope is that the contradictions that dog these traditional techniques can be explained by a completely new view. In particular, the "Dynamic Systems" approach is gaining support in Cognitive Science circles [34], [37], [4]. The following sections hope to introduce the reader to this approach and explain its take on the issue of learning.

What is Dynamic Systems?

In its simplest form, the dynamic systems approach looks at "things" (animals, ecosystems, societies, etc.) in terms of their interaction with their environment over time. Also of fundamental importance is the recognition that understanding how a whole system works does not come from analysing the functional details of the individual subcomponents of the system: rather, useful understanding comes from seeing how the subcomponents interact.

A few concepts relevant to learning systems are key parts of the dynamic systems literature. Autopoesis refers to the process of self-organization, which is a vital part of maintaining a stable structure in a complex and changing environment. Maintaining internal stability is a major reason why living organisms need to adapt (for example, food is required to maintain energy levels in a creature, so it must be able to find food in various different and novel circumstances). The concept of autopoesis also keeps in mind a fundamental rule of physics: the second law of thermodynamics. This law, traditionally ignored in cognitive science, says that in order to maintain an ordered structure, it is necessary to produce disorder in the system's environment. This sort of argument can be taken to the extreme point of arguing that the very purpose of life is to produce disorder in its environment.

A second useful concept from Dynamic Systems is the idea of feedback. In this area, dynamic systems shows the side of itself sometimes referred to as Cybernetics: originally the study of feedback in control situations. The basis of this concept is that feedback loops are vital to any control task, and the very presence of this feedback significantly complicates standard analytical mathematical techniques. Feedback generally leads to non-linear differential equations, which do not lend themselves to simple analysis or to linear approximation. Thus, a different approach to analysing such systems is needed, since linearity is a basic assumption in the majority of such techniques.

In its simplest form, this new approach (from classical systems theory) focuses on identifying positive and negative feedback loops within the system. Instead of mathematically solving the system, we look for behavioural generalities. Positive feedback loops are situations where if one aspect of the system increases (or decreases), it affects the rest of the system, which in turn ends up further increasing (or decreasing) the original aspect. This process continues, with that aspect increasing (or decreasing) rapidly. This is not a good thing for a structure that is trying to be self-organizing. This is why the analysis of most living systems tends to result in the discovery of negative feedback loops instead. These loops tend to maintain various patterns in the system. The classic example in biology is homeostasis (the ability to self-regulate various things such as body temperature). One of the recurring themes of dynamic systems is that it is studying the loops of interaction between various components that gives the greatest insight into the functioning of a system, rather than a linear "input-processing-output" analysis.

The third Dynamic Systems concept to be examined here is epigenesis. This is the "process of development" or "how the system got to be where it is now". This emphasises the temporal nature of systems. The claim is that we can understand better how a system works now by looking at how it came to be this way: at its history. This puts even more importance on the issue of development and learning.

Fourthly, there is the field of stability theory (also known as catastrophe theory, and tightly related to chaos theory). Here, the idea is that systems fall into certain natural rhythms based on their structure, and they will maintain these rhythms in the face of minor changes in their environment. However, if changes build up over time, the system will be disturbed enough to jump to a new rhythm. These "state-space transitions" between different "attractors" can take place extremely suddenly, even if the environmental change just preceding it was very small. Most importantly, once the system has jumped to the new attractor, it will be inclined to stay there. It will not naturally fall back to the original state, even if its environment has returned to its original state. The properties of the system are thus not defined purely in terms of its structure and its environment, but also in terms of the system's history.

Infant Development

The clearest example of how the Dynamic Systems approach to understanding learning differs from the traditional approaches is a look at human infants learning to walk. Nativism claims that humans have an innate ability to walk, and as an infant's muscles grow stronger and more controllable, this innate ability is simply found. This is often tied in with the idea of Central Pattern Generators (CPGs): groups of neurons that simply fire in a regular sequence, and this sequence of firings somehow controls the leg movements to produce the regular patterns observed in walking behaviour. This is basically an argument that the ability to walk is pre-specified in the genetic code itself.

Constructivism takes a slightly more complex view of learning to walk. It argues that an infant passes through certain steps along the way to learning to walk, and the skills and abilities learned at the previous steps are built upon to reach the next step. First the infant learns synchronized kicking of both legs together (or perhaps there is innate knowledge of this very simple skill). Eventually, it learns to kick out of phase. When flipped over, this sort of action can lead to learning to crawl, and so on and so forth until the infant learns to walk.

The Dynamical Systems hypothesis, however, says that both of those methods are much too rigid to explain what actually happens in human development. These specific steps that constructivism argues for may be observable when one looks at a general, overall perspective of development, but when actually looking at development in one specific individual, these rigid stages are not clear at all. In some individuals, stages may be skipped, or developed concurrently. There may even be periods where the infant slips back into previous stages. Specifically, the dynamic systems approach strongly argues that we should not be looking at stages for the "average" individual: each individual follows their own path, as determined by their interactions with their environment, for learning to walk [34].

Or, indeed, for not learning to walk. In [30], a number of examples are given showing that such behaviours as using walking for locomotion may not arise, depending on the infant's environment [7]. The other examples include the ability of people born without arms to use their legs and feet to perform tasks that are normally done with the hands. It is difficult to see how either constructivism or nativism can explain the infants learning a "scooting" behaviour instead of walking. The key aspect that dynamical systems provides here is the focus on the interaction with the environment. Without this complex interaction, learning does not take place. The reason for the appearance of a regular progression of development is that as the infant's body grows, the dynamics of the body-environment interaction gradually change.

Sensorimotor Coordination

Dynamic systems also strives to get as far away as possible from the "Input, Processing, and Output" approach. This is typified by having a system which uses sensors to gather information about the environment, then performs some processing on that information (and upon whatever information the system may have stored previously), and then performs an action. This sort of approach (and the associated "computational metaphor") has become so common that it is sometimes hard to think that it could be any other way.

However, there is a vitally important step that is left out in this mind-set: the role of the environment. The environment responds to the output and produces new inputs to the system *that*

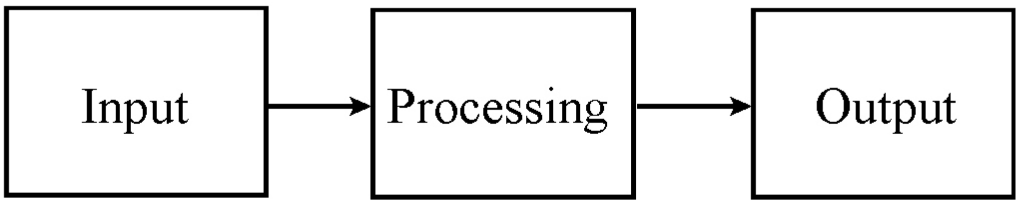

Figure 2.1: The traditional view of cognitive processing.

are based on the system's outputs. There thus is feedback in the system, which opens the door to an entirely new way of looking at it.

Initially, this feedback "revelation" may not seem to be very impressive. After all, is it really fundamentally new? It's not like all the previous researchers didn't notice that the output affects the input. What's the big deal about focussing on it like this?

One classic example to show that the feedback aspect is vitally important to keep in the forefront of one's mind is a feature of insect walking that had confused researchers for quite some time. In order for an insect to walk, there has to be some high level of coordination between the legs (especially on rough, complex terrain). The insect needs to know things like whether or not each leg has a solid grip on something, or if the ground under one foot has suddenly disappeared, and so on and so forth. However, the speed with which the insect is able to react to these sorts of situations is much faster than any signal seems to be able to be transmitted through the insect's very simple nervous system. How can it possibly do this?

The answer is, of course, feedback. What happens when the ground under one leg starts falling away? The insect starts to tip, of course. This *immediately* causes the pressure on the other legs to change. The information is thus transmitted without the insect having to do anything at all [6].

The Dynamic Systems approach maintains that this sort of use of the environment is the norm, rather than the exception. The world itself is its own best model, and it is likely that a successful

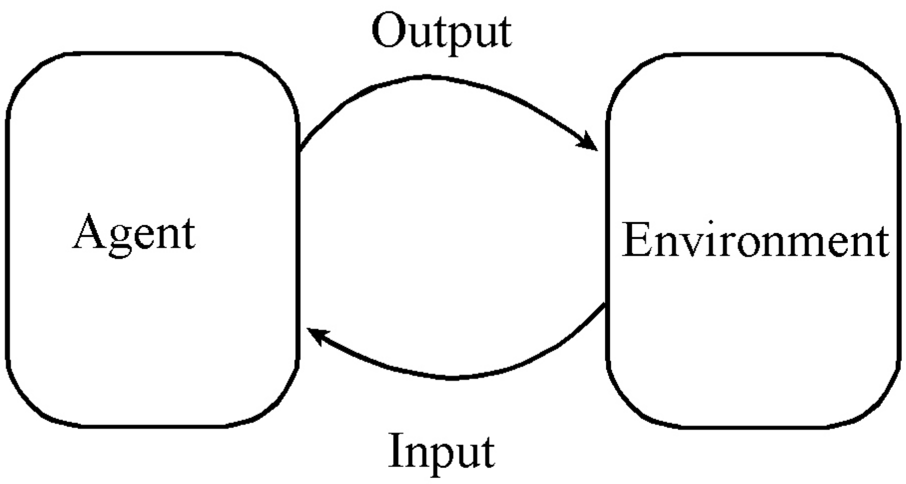

Figure 2.2: The dynamic systems view of cognitive processing.

creature will make use of this fact whenever possible.

The other key difference that focussing on sensorimotor loops produces is that it is not necessary to find the perfect answer right away. The creature is in constant interaction with the world, and so does not have to produce the perfect output for any given input. An example will make this clear. The Braitenberg Vehicle 2A [5] is widely considered to be the simplest possible way of having a robot move in an interesting way. It consists of two sensors (one on the right and one on the left) and two motors (left and right as well) that drive two wheels. The sensor on the left directly controls the motor on the right, and vice versa. Thus, if the sensors are brightness detectors, then the robot will turn towards any light source and drive towards it. If the sensors are range-finders, then the robot will run around avoiding walls. It does not seem possible at all to get this sort of behaviour from any simpler system, and indeed this architecture is very appealing in its elegance.

This was true until researchers at the University of Zurich decided to make use of the creature's interaction with the environment. They were able to produce the same behaviour using only one sensor and two motors. They made use of the fact that the sensor value would be changing over time in response to the system's interaction with the world. If the sensor value wasn't changing much, then the robot probably didn't need to turn much. If the sensor value was getting smaller (i.e. the robot was getting closer to a wall), then this is simply a sign that the robot is turning the wrong way, and it should change directions. Thus, the robot would, when it encountered a wall, turn one way or the other somewhat randomly. If this turned out to make the robot's situation worse by bringing it slightly closer to the wall, then it would change directions. This would happen so quickly that it would be unapparent to an outside observer, and so the behaviour would seem to be identical to the more complex Braitenberg system [20]. The success of this simpler control system shows that making use of the environment in this manner is part of the key to developing the efficiency and robustness found in nature.

A further issue that arises when looking at sensorimotor loops is their level of complexity. This is the main focus of [19], in which Keijzer states that "the behaving system needs complex behavioral capacities at a proximal scale – call it room for manoeuvring – to counteract the many proximal disturbances that will occur in a natural environment. Only by bringing into play many different options for proximal behavior – that is when the sensory-motor system has many degrees of freedom – can stable distal regularities be ensured under natural circumstances." In other words, a complex sensorimotor system is required to deal with a complex environment. This is a direct result of Ashby's Law of Requisite Variety: "Only variety can destroy variety" [1]. Keijzer goes further in his paper and argues that by using simple sensors and motors we do not gain insight into how an organism can deal with the complex sensors and motors that are required. In particular, he focuses on the difference between using wheels for locomotion rather than the complex musculature and skeletal system that animals use. His recommendation is to stop viewing actions as a simple unit (such as "go forward"), and rather to look at the entire sensing and acting systems that interact over time to produce something that we can then describe as "moving forward". The actual behaviour of a creature that is "moving forward" is incredibly complex, never the same twice, and is highly dependent on the agent's current environment. This dependence is an important part of how an organism can function in a complex environment, and ignoring it may significantly limit research in the field.

Another strong supporter of the sensorimotor view is [24]. Pfeifer's paper contains numerous citations arguing that "classification, perception, and memory should be viewed as sensory-motor coordinations rather than as individual modules." Many of these references are also made use of in this document, especially [34] (see section 2.4.2) and [9] (see section 3.3). Furthermore, a significant amount of work is being done on this issue at Pfeifer's AI Lab at the University of Zurich, and so the curious reader is directed there.

Values

One vital aspect of development that has not been mentioned as of yet is what actually directs the development. How does the system choose what to change about itself? This is an incredibly complicated question, and is, indeed, at the very core of any discussion of learning. Different approaches to answering this question lead to completely different development architectures.

Sometimes this thing which directs development is referred to as a goal. However, this choice of word seems to imply that there is some sort of ideal end-state towards which the system is moving. This sort of approach is completely opposite to the dynamical systems approach, which focuses on the path the development takes. Therefore, the term “values” is often used, as it deals with how the system wants to change at a particular time, and in a particular situation. Unfortunately, the word “value” is also loaded with connotations like those of “goal”, which can still lead to some confusion. It is hoped that the reader will keep in mind that “value” here refers only to the basic, immediate rules governing the organism.

Values, then, are the low-level, generally unchangeable, aspects which drive the development of a system. We can see them as being specified by the genetics of the creature, or the initial configuration and rules of a complex system. One of the startling and non-intuitive results of both Artificial Life research and Dynamic Systems research is that very simple values can result in enormously complex and unexpected behaviour. This is referred to as *emergence*.

One common example of a value used in learning is regulation. At the most basic level, living creatures need to maintain certain features of their environment at a regular level (such as body temperature, or the availability of food). We have already mentioned the phenomenon of homeostasis in terms of a negative feedback loop (see section 2.4.2), where a creature’s body will automatically initiate actions such as shivering when cold to maintain body temperature. If we extend this concept further, we can look at learning as a highly complex negative feedback loop which allows us to adjust high-level behaviour in order to maintain regularities. This point of view offers a solution to one of the major questions in the field of learning: what exactly should we learn? Why do we learn one thing and not another? To give an extreme example, why don’t I learn to stand in a corner hitting my head against a wall? The answer, or at least part of the answer, is that we learn things that help us to regulate our environment.

For further information, [31] gives a broad range of examples of the use of value systems in both natural and artificial systems.

2.5 On Concepts and Existence

Before going further, I would like to take a moment to discuss the relationship between concepts and “existence”. While this is not a thesis focussing on philosophy, I do feel that it is relevant to describe the approach I take on this issue, because it is fundamental to the direction that this thesis takes.

I would like to talk about existence in one of the simplest frameworks possible: Conway’s Game of Life [11]. This is a standard cellular-automata system, where each square in the grid can be in one of two states: full or empty. At each time step, the state of each square is changed according to two simple rules: if a full square has less than two or more than three full neighbours (including diagonals) then it becomes empty, and if an empty square has exactly three full neighbours, it becomes full. Generally, the squares in the world are put into a certain state, and then the system is left to run for a while on its own.

Some experimenting with this very simple world can lead to some very interesting results. Certain patterns seem to come up frequently. For example, a 2 by 2 square is stable and, once formed, does not change until something else comes near it. A line of 3 full squares will continually alternate between horizontal and vertical (until it is disturbed). Another remarkable structure is the glider, where a particular pattern will actually move across the world. There are even complex systems referred to as glider guns which will actually create an infinite number of regularly spaced gliders.

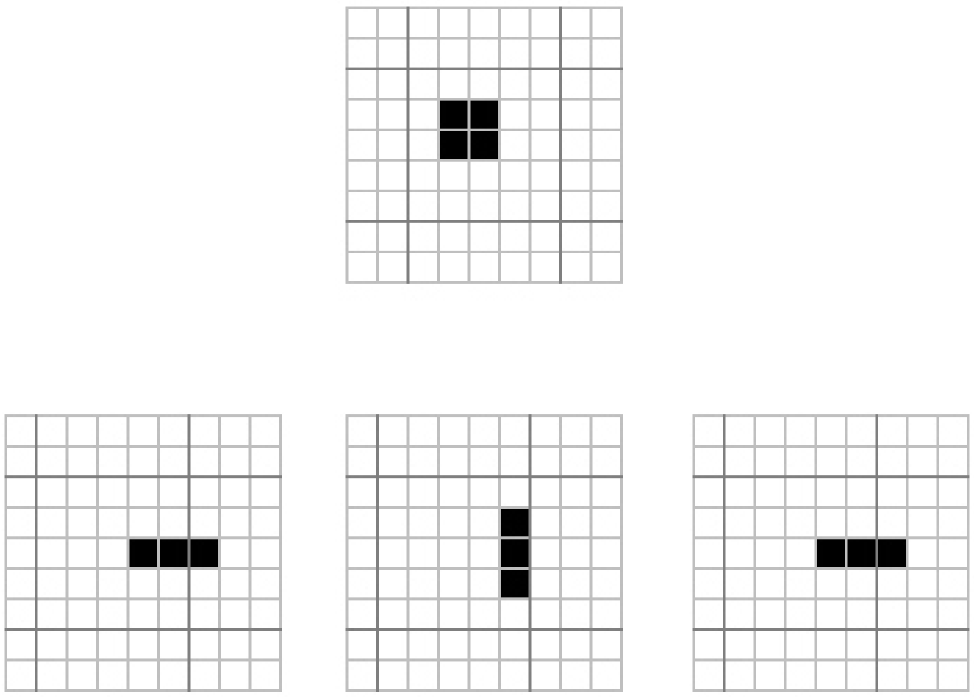

Figure 2.3: Two simple stable patterns in Conway's Game of Life. The square does not change over time, and three cells in a horizontal line will cycle between horizontal and vertical ad infinitum.

Now, let us consider what "existence" means in this world. It seems clear that the squares in the grid do, in fact, "exist". Everyone can agree on what defines a square, it has a particular location, and it has particular states. What I am interested in, though, is the following question: does the "glider" exist?

I would argue that there exists a point of view from which the glider does not exist. We may call this point of view the "objective" point of view (although "objective" tends to imply

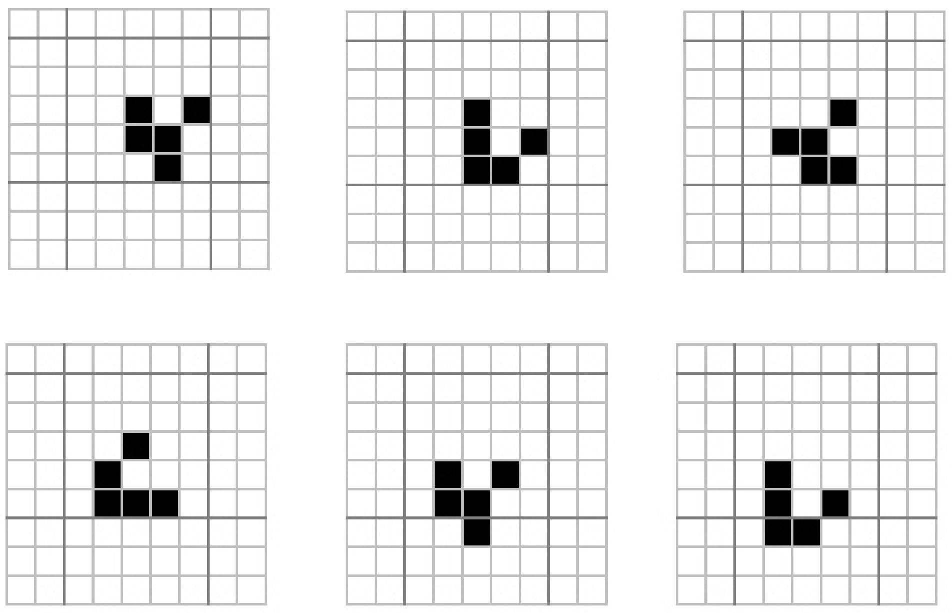

Figure 2.4: The classic glider. Over the course of four time-steps, the glider moves one square down and one square left. Images thanks to Mirek's Cellebration [46].

"correct", which is not my intention here), or even "the world's point of view". As far as the world is concerned, all that exists are grid points, states, and rules for determining the next state. This is a completely objective and universal point of view, that can be considered to represent the objective "Reality". However, it is true that if a glider is shown to almost any observer, they will immediately recognize it as a "thing". It is a regular, observable pattern. We can make predictions about it. We can talk about its properties like speed and direction. Importantly, it can be independently discovered by different people.

However, it must be noted that referring to that pattern as a glider is not always the best way to refer to it. When the glider interacts with other full grid squares, what happens is best described and understood by referring to the squares themselves. Depending on the configuration of the squares the glider is running into, the glider may "cease to exist". I would note, however, that it is a pattern that is ceasing to exist, not something "Real".

I hope that the analogy to the actual world is clear. I would argue that something like a "table" does not "Really" exist. It is a pattern in whatever the underlying "Reality" is. However, we can form predictions about it. We can talk about its properties. We can all generally agree that it is a "thing", and, indeed, it is useful to call it a "thing" in most circumstances. However, in extreme situations, we cannot in fact achieve a consensus on certain traits the table may have. For example, is a scratch on the table part of the table? In fact, is a scratch itself a "thing"? If something stains the tabletop, is that substance now part of the table or do we now have two things intermixed in space?

The important point in all of this is to note that pretty much everything that we refer to as "existing" is actually a pattern, and as such, is subjective. Different people, with different backgrounds, find different patterns in what may be the same underlying Reality. This is a sort of subjectiveness which is completely apart from the subjectivity caused by different observers receiving slightly different data. This is a subjectivity caused by different world-views. Patterns are identified as "regularities in our environment that we observe over time", and since individuals come from different environments, they will find different regularities in the exact same data.

If people are raised in the same culture, and thus have very similar environments to find patterns in, they will generally form similar patterns. These people can then communicate to each other reasonably well, since they will be able to connect various sounds (words) with various patterns (or "concepts") in a relatively similar manner.

Sometimes, however, we will find difficulties where a pattern we have seen is not a pattern that the other person has seen. In this case, the only way to achieve communication is to expose the other person to situations where that pattern exists (through actual experience, or perhaps education and imagination), and allow them to identify that pattern.

I believe that this is how we form "concepts". We observe regularities in our environment. People with similar experiences find similar regularities. I expect to see a similar process occurring in artificial life. The learning algorithms I will discuss in this paper will be looked at in terms of their pattern-finding and pattern-using capabilities.

Chapter 3

Learning in Artificial Systems

3.1 What are Artificial Systems?

For the purposes of this document, I am taking “Artificial System” to mean any software program that takes in a known, finite set of input variables and produces a known, finite set of output variables (and modifies its internal state) based on its internal state. This is the basic definition of a Turing machine, and as such covers a lot of ground. (It is assumed that the reader is familiar with software programs and the concept of a Turing machine). To narrow this a little, I wish to focus on connectionist systems (also known as Artificial Neural Networks).

Artificial Neural Networks consist of large numbers of very simple components connected together. The general design is somewhat inspired by observations of the nervous systems of living creatures, which are formed from highly interconnected neurons. It must, of course, be remembered that a single real neuron is much more complicated than any component implemented in an ANN. However, like the real neurons, the artificial neurons (also called nodes) have inputs and outputs, and the output is formed by combining the inputs together in some way (generally summing them) and then performing some function on that value (such as a sigmoid function). Connections can be excitatory (positive) or inhibitory (negative), and generally each connection has a weighting factor that indicates how strong a connection it is.

This sort of artificial system has three important advantages over other sorts of software systems, such as classical AI’s knowledge-based systems. First of all, they allow for parallel, localized computing. Secondly , they are actually neurologically plausible. Thirdly, they tend to exhibit the sort of generalization and simple learning capabilities that are expected.

We have not yet discussed how exactly these neurons are to be connected, nor have we discussed how connections between neurons (or the neurons themselves) change and develop. These factors are what distinguish various different ANNs from each other. As this section progresses, we will examine a few relevant systems developed by various researchers.

3.2 Classical Conditioning and Association

Classical conditioning is, in essence, the forming of associations. Initially, a certain stimulus (say, the presence of food) is associated with a certain response (salivating). The experimenter then presents a new stimulus (the ringing of a bell) at the same time (or around the same time) as the initial stimulus. After a few trials, the subject will give the same response when presented with the new stimulus by itself. This sort of behaviour is found in pretty much all animate life forms, including simple things like planarian worms. There is, of course, the question as to what associations are being formed. Is the new stimulus being associated with the response? Or is the new stimulus being associated with the old stimulus, so that when the new stimulus is presented, it is as if the old one is presented? And furthermore, what rule forms these associations? These questions are the core of behaviourist psychology.

It must be noted, however, that behaviourist psychology traditionally only looks at associations between stimuli (input) and responses (output). It has become increasingly clear that this is a much more limited domain than most living creatures use. For example, creatures can associate actions with certain *places*, and a place is much more complicated than just a stimulus [3]. It is clear that there are intermediate things between stimuli and responses, such as internal states, that can also form associations. This concept will be further developed in the section on the Theory of Neuronal Group Selection (see section 3.3).

The simplest rule for forming these associations is Hebb's rule. This rule is, very simply, that the connection between any two components in a connectionist system should be strengthened if both of those components are active at the same time. It is clear how a rule of this form can explain the classical conditioning results in the example at the beginning of this section. The stimulus that the bell is ringing would activate a node in the network at the same time as the presence-of-food node is active. Initially, the food node is strongly connected to the salivation node (perhaps by genetic design, or perhaps by previous learning), and so that node would become active. Now, the bell ringing node is active at the same time as the salivation node, so the strength between them would increase. If this happens a few times, then the new connection will increase in strength until eventually it is able to activate the salivation node all by itself. Note that, most likely, all sorts of other nodes representing other stimuli and responses (and other, internal states), will also be active while this learning is going on. For example, there might be a node active if the experimenter is wearing a red shirt. If this is active while the feeding is taking place, then an association will start to form there. However, it will only become a strong association if it is a regular occurrence. Thus, we can see that this sort of association forming is actually an identification of regularities in the environment.

One of the more complex models of conditioning can be found in [26]. This model accurately describes the acquisition of an association, the extinction of that association, and the effect of blocking. However, it does this by assuming that the stimulus has already been categorized into different types of events. That is, there is no sense of "similarity" between stimuli. This means that it is incapable of handling the generalization of learned responses to new situations, and avoids the entire problem of determining when two different events are actually instances of the same stimulus.

More details about the various aspects of classical conditioning will be examined in a later section, where we examine the Distributed Adaptive Control model (see section 4.3). For now, however, it is important to remember that classical conditioning includes the ability to un-learn associations that are no longer valid, and to both generalize and learn to discriminate between initially similar patterns. All of these features are necessary for a creature to deal with a complex and changing environment.

3.3 Edelman's Theory of Neuronal Group Selection

The Theory of Neuronal Group Selection (TNGS) is a model of general learning developed by Edelman (along with Reeke and others) that explicitly deals with a number of the issues that have been discussed so far in this paper. It focuses on being a neurologically plausible explanation of learning, and exhibits some very interesting properties, such as multimodal categorization. The key book which details this theory is *Neural Darwinism: The theory of neuronal group selection* [9]. Of particular interest is the fact that it incorporates both a developmental phase and a learning phase, which I have previously argued to be important (see section 2.1).

In order to describe the theory, I believe two different approaches are necessary. First, I will describe the theory from a developmental point of view, and then I will describe it from an architectural point of view. After this, I will discuss some comments and criticisms of the theory.

3.3.1 TNGS: A Developmental Perspective

The inspiration for the theory came from Edelman's 1972 Nobel-prize winning research on the immune system. His work argued that the immune system uses a process very similar to Darwinian selection to develop antibodies for new foreign substances that enter the body. The process is very simple: large numbers of slightly different, randomly mutated antibodies are created, and those that are successful are duplicated while those that are not successful are not. Thus it is an example of using the "blind-watchmaker" (design without a designer) concept on an immediate, short-term time scale, rather than at the level of evolving species, as it is normally thought of in Darwinian evolution.

With the TNGS, Edelman argues that this same process can be used to develop intelligence. He starts with the observation made earlier in this document that the human genome is unable to explicitly define all of the complex structure of the brain. What is defined, however, is a basic framework for development. The question is what form this framework takes, and how this development occurs.

This question may lead one to believe that the DNA defines the initial state of the brain before developmental learning occurs. Edelman specifically argues against this point of view. His theory holds that the vast majority of the neurons in the brain are configured randomly. The overall structure of the formation of neurons may be somewhat controlled via various chemicals, but this is still a highly chaotic and non-deterministic process. The neurons (in the vast majority of the brain) not only have highly varying and uncontrolled properties, but also connect to each other in a similarly chaotic fashion. This aspect of the theory is supported by the observation that, while we may be able to find common high-level structures in different people's brains, there is a huge amount of variation at the neuronal level (both between individuals and within a single individual).

Thus, we start life with a pretty much random mishmash of connections of neurons in the brain, separated into basic functional areas. How, then, does this become a useful system for controlling the complex actions that humans do? Edelman's answer to this question is "selection". Because of the huge numbers of neurons in the brain, this random approach guarantees a huge "primary repertoire" of functionality. Some of the connections between the neurons form useful systems. Some do not. The connections that are "useful" (as defined by some pre-determined "value" system (see section 2.4.2)) are strengthened. This process forms a large number of strongly connected "neuronal groups".

Of course, how we define "useful" is always a difficult question. It is, in fact, likely that there is no one answer to the question, and that it depends on what sort of system is being controlled by the neural groups (i.e. what region of the brain the neurons are in, or what sensor and motor groups they may or may not be connected to). The process could be as simple as "strengthen those connections which were used within a few minutes of feeling happy" (which could, perhaps, be implemented through synapses that vary their strength based on mutual activity and the presence of various hormones). It could also be much more complex. However this is accomplished, the theory states that the first developmental step is in creating this "secondary repertoire" of neuronal groups.

There is now, however, the question of how these groups come to be connected to one another. Edelman postulates that, as these groups are forming, there is also a process of "reentrant mapping" going on. This is the creation of connections between neurons in different groups based on their statistical correlation to one another. This is the aspect of making "associations" that was discussed earlier (see section 3.2).

The following diagram shows the three major processes at work in TNGS.

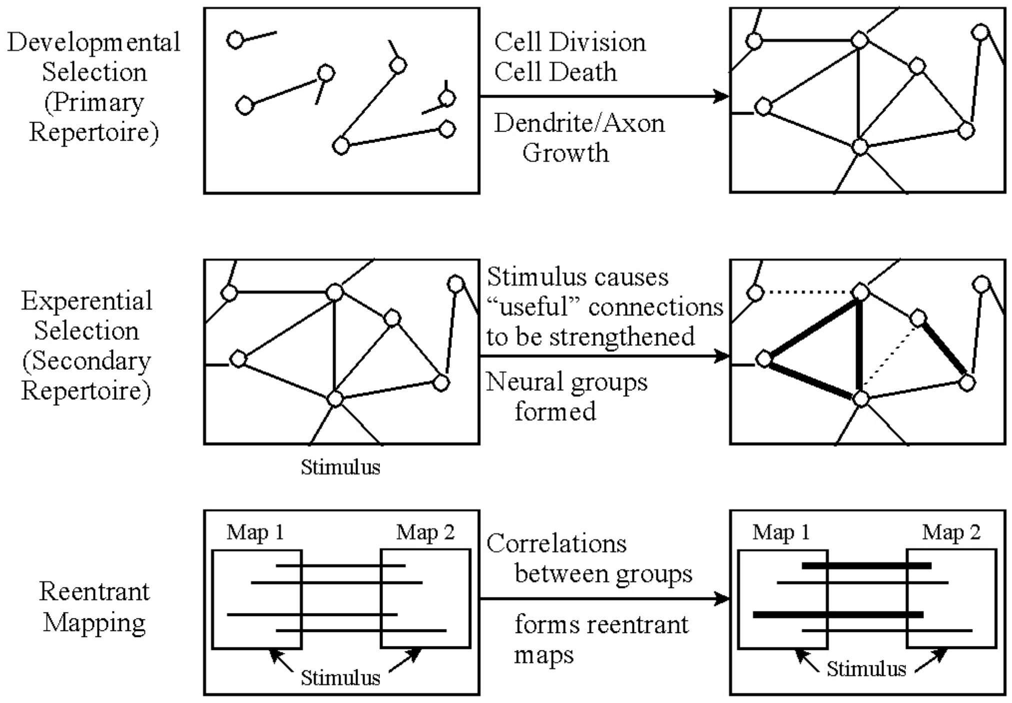
Developmental Selection (Primary Repertoire)
Cell Division
Cell Death
Dendrite/Axon Growth
Experential Selection (Secondary Repertoire)
Stimulus causes "useful" connections to be strengthened
Neural groups formed
Stimulus
Reentrant Mapping
Map 1
Map 2
Correlations between groups
forms reentrant maps
Stimulus

Figure 3.1: The three major processes of TNGS. First, the growth rules of neurons creates a large number of interconnecting neurons. Next, connections between the neurons are strengthened and weakened following built-in rules ("values") based on the received stimulus, forming a number of neural groups. Finally, mappings are formed between the groups based on the correlation of their outputs given the same stimulus. These three processes may all occur at the same time. Based on Figure 3.1 in Edelman, 1989

3.3.2 TNGS: An Architectural Perspective

Another way to look at the Theory of Neuronal Group Selection is from the perspective of the final developed architecture. In this way, we can try to understand what it does once the development is mostly complete. Of course, we must remember that the process is always continuing during life (thus allowing us to learn new things). However, an architectural view will allow us to compare it to other approaches to Artificial Life, as most other approaches focus on the architecture, rather than the developmental (or epigenetic (see section 2.4.2)) process.

The basic architecture is a map of maps of maps. Recall from the section on classical conditioning (see section 3.2) that one approach to "learning" is to form associations (or mappings) between inputs and/or outputs. One issue that the discussion in that section revealed is that this was not sufficient, and that associations needed to be able to form between internal state variables to give the range of behaviours that we observe in living creatures. TNGS deals with this in the following way.

First, we note the sensors and motors of the agent are in fact complex mappings to and from the environment. For example, the information received by the eyes is not a nice three dimensional breakdown of the world: instead, it is two dimensional data, distorted by perspective, with much more detail near the centre of the field of view than the edges. There isn't even straight-forward colour information available, since there are three separate types of colour sensors in the eyes, each responding in differing ways to various different frequencies of light. These sorts of issues are true for all but the simplest types of sensors and motors available to an agent, since they are generally constrained by the physics of the agent-environment interaction. Why should we believe that this raw data is suitable data on which the agent can create associations?

This question is, of course, nothing new. This is the sort of issue that is generally dealt with by pre-processors and post-processors, systems which massage the data into a more useful format. In robotics, for example, it is quite common for the AI controller to specify an output like "rotate the arm to an angle of 65.2 degrees", and then a post-processor would translate that into a specific voltage to apply to the motor on the arm for a specific period of time. Another example would be the postulated range-finders used quite commonly in Artificial Life experiments, and mentioned in relation to the Braitenberg Vehicle in the section on sensorimotor coordination. A real-life range-finder is remarkably complicated, and has to do all sorts of processing in order to finally output a value that says how far away the nearest object in a certain direction is. All of these are examples of mapping the raw sensor data (which is itself a mapping of environmental data) into new data, which is, in theory, a more useful value for the rest of the system.

One important point that TNGS makes is that this second mapping should be done more than once. Why should we look at the raw data in only one way? Why not look at the same data in many different ways? Let us have multiple, completely independent mappings of the raw data. In one early experiment following the TNGS philosophy, Edelman and Reeke worked with a raw sensor that looked at letters of the alphabet (drawn in various different styles). They then created two different, completely independent mappings of this raw data. One mapping identified various spatial features of the letter, and one identified various temporal features (i.e. how the letter was drawn). Both neural groups received the same raw information, but the structure of one group was such that it extracted temporal information, and the structure of the other was that it extracted spatial information.

We now create mappings between the mappings themselves. In other words, we form associations between the outputs of these secondary mapping systems. The real issue here is, how exactly we would go about doing this. With TNGS, we allow the agent to experience the world, and while it does this, we strengthen connections between correlated results of the different maps. In other words, we form correlations between different "aspects" of the sensory data that are active at the same time while we explore the world. This lets us see which of these randomly formed mapping systems is likely to be a meaningful representation, since meaningful data is generally correlated to other meaningful data. Furthermore, it gives the system all of the capabilities of the associative systems discussed earlier (see section 3.2), but with the significant advantage that the correlations do not have to be made on the raw sensory data.

TNGS thus results in a map of maps of maps. The practical result of this can be very interesting. In the experiment mentioned above with the spatial and temporal features of letters, the internal mapping resulted in a system that was able to learn to categorize letters, based only on being presented a large number of different letters drawn in very different font styles. The important point is that it was never told that there was a distinction between the 26 different letters (i.e. it was never explicitly told that **A** was an A and *A* was an A, but *B* was a B). Simply by observing the correlations between the spatial and temporal features, the system formed 26 different significant statistical correlations, with one corresponding to each letter. Importantly, it did this without even being told that there were exactly 26 different patterns to find.

This example is a very simple one, as it deals with a system with only one sensory modality. TNGS also works in situations where there is a more direct interaction between the agent and the environment. This has been done in experiments around Darwin III, a robot arm that uses TNGS to learn to follow moving objects.

3.3.3 Comments

A few aspects of this theory strike me as being important for consideration. The first of these is the scale at which the system works. Edelman's use of this architecture in a model of the mammalian visual system uses 220,000 artificial neurons and 8.5 million connections between them [9]. Current research in neural networks is seldom seen at this scale. Most neural approaches to intelligence take the philosophy of directly specifying the majority of connections in the model, and then use some method to modify the strengths of those connections. However, the standard methods for doing this (such as genetic algorithms or back-propagation) do not seem to easily scale up to large systems. In contrast, TNGS actually *requires* large populations of neurons, since that is how it achieves the diversity of functionality that is necessary for interesting behaviour. Thus, it may be that techniques which work on small neural networks and less complex environments are unable to work on the level of complexity that I am interested in. Conversely, it is likely that TNGS will have very little to say about simple nervous systems composed of merely dozens of neurons. It should also be noted that TNGS is fairly unique among learning algorithms in that it has no pre-specified architecture. Recalling the discussion on why learning is necessary (see section 2.2), exact specification of the connections in the brain is unlikely to be possible in a system as large as the brain.

This is important in that it takes a concept from the "New AI" (a.k.a. "Nouveau AI") approach even farther. "New AI" researchers generally hold that in order to create "intelligent" systems, it is not feasible or even possible for a human designer to specify everything about how the system works. That sort of direct, rigorous programming of behaviours is often seen as the reason why "Good Old-Fashioned AI" was unsuccessful at producing systems capable of interacting in a complex environment. Instead, New AI researchers have taken approaches such as defining a structure and having the system modify the connections between the parts of the structure through experience. What TNGS argues is that we can take advantage of the larger number of neurons in a large-scale system by not specifying the architecture at the low level.

A second major point of interest is one that I have not seen specifically mentioned in discussions of TNGS, but is one which may perhaps not be obvious to many people reading about the architecture. I present it here simply as a comment on the capabilities of the system. It concerns the function of the "secondary repertoire" (the neural groups formed somewhat randomly between which the correlations are made). These neural groups are generally thought of as "preprocessors" or "filters" on the incoming sensory data, which, in theory, put the data into a more useful form. While this terminology is accurate, it causes people to think of these filters as being time-invariant; that is, given the same input, they will always give the same output.

This is not necessarily true. Many of these neural groups will have complex, recurrent connections between the neurons, forming loops and complex structures. These loops imply that the neural groups can fairly easily have a memory, and can respond to patterns formed over time, not just to patterns among the sensory data at one instant in time. This is because the output of the group will be based not only on the inputs to the group, but also on the state of the group at the previous moment. For example, in the following diagram, the output from the network will depend not only on the input, but also on the current state of the lower neuron. This is a general property of any network containing loops.

Finally, it must be mentioned that the Theory of Neuronal Group Selection has great intuitive appeal. It is another example of the "brute-force" general problem solving method used to great success in natural evolution. The idea of trying lots and lots of different possible approaches all at the same time is simple, and yet it is also the exact antithesis of the formal, step-by-step logical approach that represents most of scientific thought. Of course, it could be argued that the reason that the scientific approach has been so successful is that there are large numbers of scientists working in parallel, each with their own unique (and somewhat random) ideas and perspectives, with an organized structure to allow them to find correlations between their work and others'. In any case, TNGS applies this approach to the development of intelligence in a way that seems possible, given what we know about brain structures and how neurons form. We know that controlling the exact formation of neurons and connections between them is remarkably complicated and perhaps impossible in many situations. It seems somehow right that we should embrace this randomness and make use of it, rather than fighting it.

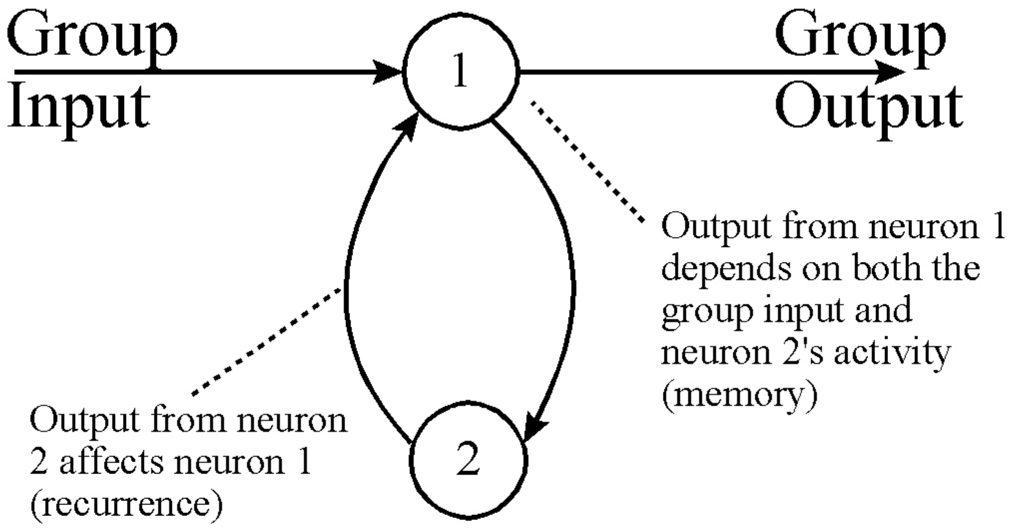
Group Input
1
Group Output
Output from neuron 1 depends on both the group input and neuron 2's activity (memory)
Output from neuron 2 affects neuron 1 (recurrence)
2

Figure 3.2: A simple network showing how recurrent connections (connections forming loops) lead to memory. The output is dependent not only on the input, but also on the state of the lower neuron.

3.3.4 Criticisms

There are, however, two important problems with TNGS as it stands. The first of these has to do with the fact that it is a rather under-specified theory. The second issue stands as more of a warning: very few people have worked directly with the theory.

In saying the theory is "under-specified", I am making reference to the large number of unknowns within it. How exactly are the initially neural connections "randomly" formed? Edelman explicitly talks about controlling this formation in nature via cell adhesion and substrate adhesion molecules, which could generally control the size and type of the clusters that are formed. However, in an experimental situation, it is up to the individual experimenter to "tweak" these parameters, controlling growth in various ways until it works for the given situation. Furthermore, the rules regarding the further development of the network are also non-specified. TNGS is more of a framework than a specific model, and there seems to be little experimental evidence revealing what sorts of configurations are good for what sorts of problems.

This lack of general experimental results stems from the other problem with TNGS: there is relatively little work being done within the framework. Most documents describing the system seem to be vague on details, and tend not to have a clear distinction regarding what aspects are part of the theory, and what aspects are implementation-specific. This lack of use means that there exist few informed perspectives on its applicability in various situations. Furthermore, it has led to a lack of evolution of the theory, as not much has changed within it since it was first published. This lack of new data from new experiments is troublesome, even though the theory does have a few interesting and successful examples.

3.4 Pattern Discovery in Neural Networks

In the previous section, we used randomly formed "neural groups" to change raw sensory data into a (hopefully) more useful form. In this section, we will try to see whether a more direct approach would be more appropriate to the random one. After all, a great many of the randomly generated neural groups are likely to be useless. If we could specify the form of these neural groups in such a way that they would generally find useful patterns and useful ways of representing the sensory data, then this could greatly improve the effectiveness of our artificial systems.

This section starts with a general discussion of unsupervised neural networks. It will then move to a more in-depth look at the types of outputs that different networks give, and in particular how useful these representations may be for the sort of association-forming that was discussed in the previous section. Next, two particular network models will be examined, both of which seem to be exactly the sort of systems that would be suitable for use. Finally, the limitations of this more directed and controlled approach will be discussed.

3.4.1 Unsupervised Neural Networks

The world of neural networks can be, very roughly, divided into two camps. On the one hand, there is the study of *supervised* neural networks. With supervised networks, we give the system a large number of different input patterns, and for each input pattern, we tell it what the output pattern should be. Once the system has learned these examples, we can give it new input patterns that it has never seen before, and it will give some output pattern as a "best guess" as to what the output pattern should be, given its previous knowledge.

The problem with this approach is, of course, that in our situation we do not know what the outputs should be. We want the network to figure out its own way of representing the data; we don't want to impose a representation upon it. This takes us to the realm of *unsupervised* neural networks. The network is shown a large set of input data, and the network finds patterns in the data and bases its output on those patterns. The types of patterns that it finds is dependent on the learning algorithm being used.

One question that may arise is "Why doesn't the network just output exactly what the input is?" After all, this trivial output would be exactly based on the input data. Of course, this really isn't what we want. In general, we would like to have the output data be *simpler* than our input data. There is a huge amount of data coming into our sensory system every moment. It is likely that we would need it in a simpler form in order to do anything with it. So, in general with these unsupervised networks, the amount of output data is less than the amount of input data.

3.4.2 Local Representation

If we want the output data to be simple, then the simplest approach possible would be what is referred to as a classifier. In this sort of network, each output means a different "thing", and only one output is active at a time. For example, suppose the input to the network was a picture of something. Now imagine we have a number of different output neurons. One neuron becomes active if the picture is of a rock; one neuron becomes active if the picture is of a tree; one neuron becomes active if the picture is of a car, and so forth. This is referred to as "local representation" because we represent the entire picture by the activity of one neuron.

Given this example, it is difficult to see how such a system would work. After all, we are not telling the network "this picture is of a rock" and "this picture is of a tree". That would be supervised learning. Instead, we are just presenting it with a number of different pictures, and the network is classifying them into different categories that it invents itself. In most (if not all) implementations of this form of learning, these categories are based on similarity. If two inputs are similar (under whatever measure of similarity we are using), then they are likely to be put into the same category.

The following example illustrates how this process may work in practice. For simplicity, consider a neural network receiving only two inputs. These two inputs will be measurements from certain sensors: in this example, let us suppose these measurements are the size and the roundness (eccentricity) of various fruits that are represented to the system. If we plotted each fruit on the following scatter-chart, we may see something like this:

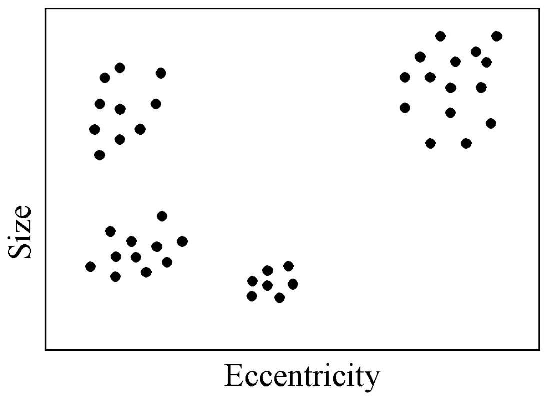

Figure 3.3: An example of the type of pattern that local representation networks can easily identify.

These points can clearly be grouped into four general categories. This is exactly what an unsupervised neural network implementing a classifier system would do. It would identify these four groups, and then whenever a new piece of data would be presented, it would output which of the four groups to which it belonged. This output would be formed by four separate output neurons, one for each category. For any given input, exactly one of the four outputs would be "active" (represented by a 1), while the others would all be "inactive" (represented by a 0). Thus, in general, the network could be given an input of two pieces of data (size and eccentricity) and would output one of (0,0,0,1), (0,0,1,0), (0,1,0,0), or (1,0,0,0). In this particular case, the group of very small slightly eccentric shapes could be limes (0,0,0,1), the slightly larger and much rounder ones would be oranges (0,0,1,0), the much larger yet still round group could be grapefruit (0,1,0,0), and the largest and very eccentric ones would be bananas (1,0,0,0).

It should be noted that it is also possible to have the network output values that are not just 1's and 0's. In this way, the network can output how sure it is that the input falls into any particular category (i.e. how similar this new input is to other examples of this category). For example, it could output (0.1, 0.2, 0.9, 0.4), which could be interpreted as saying "I'm pretty sure it's of type 3, but it might be type 4, and it definitely isn't types 1 or 2".

Even though the network has taken a two-dimensional input and created a four-dimensional output, the output is still simpler (in the sense of being easier to use) than the input. In the first case, the network has taken its two-dimensional, real-numbered input and turned it into one of four possible outputs. This is clearly a simpler format. In the second case, there are four real-numbered outputs, but it is still true that there will be *at most* one large value in the output, representing the best category match. The output data is thus much more immediately useful than the original two input values.

The exact method by which this grouping is done varies greatly between different models. The key issues involved are things like knowing how many groups are used to break down the data. Many algorithms have the user specify the number of groups, thus making it a supervised algorithm to a certain degree. Another approach would be to have the user specify the amount of variation allowed within a group. A real problem also occurs when groups are of significantly different sizes. For example, in the following diagram, point A is closer to the centre of group 2 than group 1, yet it clearly should be classified as being part of group 1.

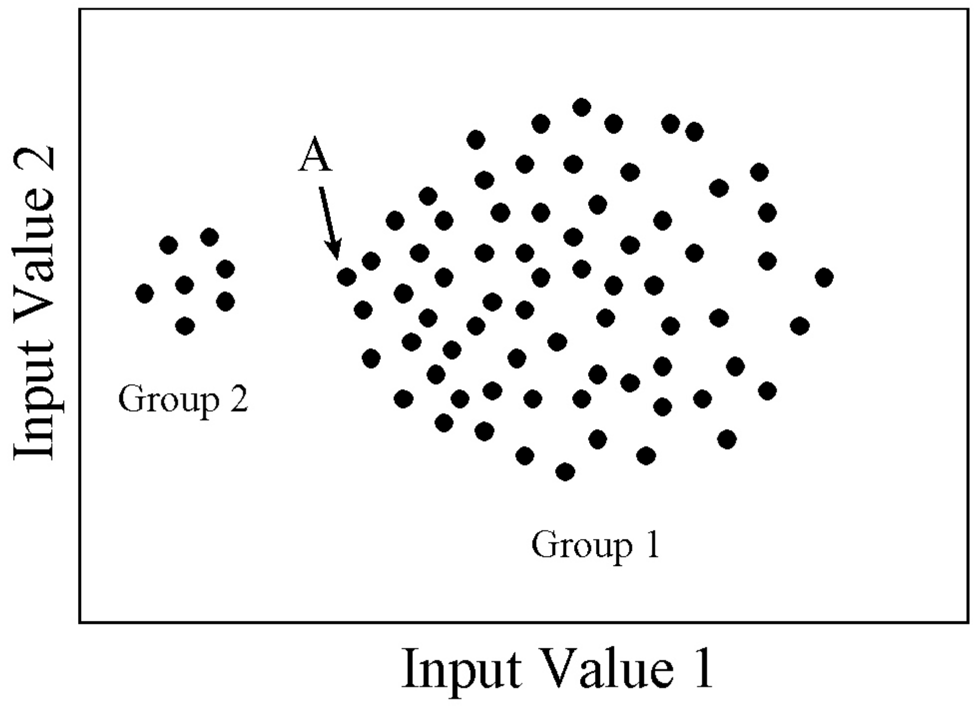

Figure 3.4: The naive approach of classifying a point by looking at how close it is to the centre of the known groups will fail in this situation.

These sorts of implementation issues, however, do not address a key problem inherent in the concept of local representation. Going back to the previous example where a network was presented with a picture of a tree, or of a rock, what would happen if the picture was of both a tree *and* a rock? Depending on the specifics of the situation, the network could just classify it as a picture of a tree, or perhaps of a rock, or perhaps it could create a new category for "tree and rock". What we would prefer is if the network would simply make both the "tree" and the "rock" output active. However, since these systems use local representation, the algorithms that they use only allow one of the output neurons to be active at a time. It seems that what we need would be a more distributed representation in our output.

3.4.3 Distributed Representation

The basic idea behind networks that use a distributed representation is to use all of the output nodes to represent a given input. Thus, instead of an output being of the form (0,0,1,0), it could be of the form (0.2, 0.8, 1, -0.5). Note that this is fundamentally different from the example in the section on Local Representation where the values indicated how confident the network was that the input was of the various different classes. Instead, these values indicate different *aspects* of the given input. To understand what this means, consider the following example where we have a two-dimensional input (i.e. there are two input neurons) and a single output.

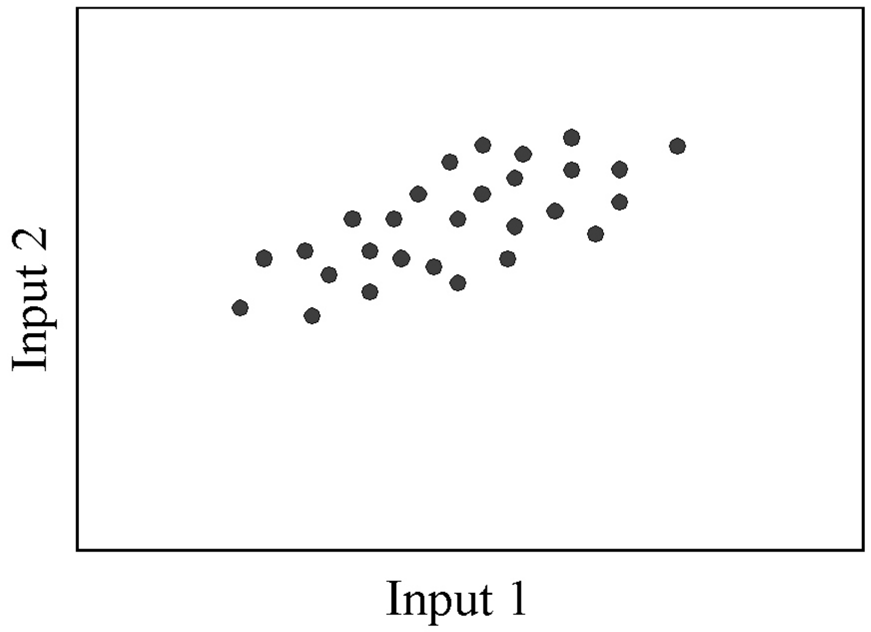

Figure 3.5: An example input data set. How can we reduce the data from two dimensions to one dimension with minimal loss of information?

Since we are reducing the number of dimensions from our input to our output, we are clearly losing information. What we want to do is to lose the least amount of information possible. That is, we want to choose the most accurate way of representing each of those two-dimensional input data points with only a single value.

The mathematics behind this are surprisingly straight-forward. We normalize the data so that the average value in each dimension is zero (i.e. we move the data so that the centre of the points is at the origin), then we put all the data into a matrix, with each data point as a column in that matrix, and we find the eigenvalues and eigenvectors of that matrix. The eigenvector with the largest eigenvalue is the best single vector to transform all of the data onto. If we want to have two output values, then we also use the eigenvector with the second largest eigenvalue, and so on.

In the example diagram above, this corresponds to projecting each data point perpendicularly onto the line in the following diagram. The output value for each point would be the distance along the line from the centre of the cluster to the projected point.

This process is referred to as Principal Component Analysis. Most importantly, the equivalent process to the algebra described above can in fact be done in a neural network. This means that the process can be done automatically and does not require the storage of all of the previous data

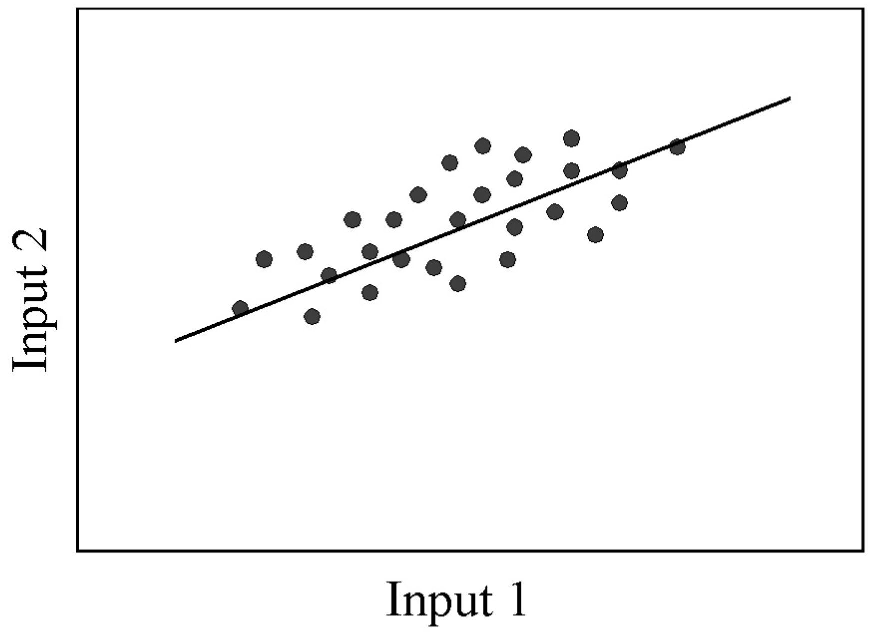

Figure 3.6: Projecting the data onto the shown line will minimize the error.

points.

It should also be noted that this process can become even more complicated. Specifically, there is also an algorithm known as Factor Analysis which begins with Principal Component Analysis and then takes it one step farther. Once the principal components (the largest eigenvectors) are known, it takes advantage of the fact that one can rotate coordinate spaces without losing information. That is, instead of taking the eigenvectors themselves, it tries to rotate them in various ways so as to produce output distributions that are uncorrelated. Being uncorrelated is an attempt to make the outputs more "useful" without losing any information. It is, however, much more complicated to implement in a neural network system. Implementations generally make use of the separation of recognition (forwards) and generative (backwards) weights, which will be discussed in the next section in the context of sparse representation algorithms.

One major difficulty with this style of analysis is that it is generally linear. That is, it assumes that all inputs are formed from Gaussian distributions. This approximation limits it to finding patterns of a particular type, and a great many useful patterns are not based on Gaussian distributions. More importantly, a fundamental problem with distributed representation in general is that it represents any given input using *all* of the outputs. This is as much of a problem as Local Representation's use of only one of the outputs for any given input. Having a completely distributed representation means that we cannot break it down into chunks; instead, we have to always deal with the entire representation. Every bit of data is important, and we have to look at all of it at the same time to retain any meaning. If we could have a representation that was not completely distributed, then we could "pick out" key aspects of the data from the full representation. This idea will become clearer with the example in the following section where we apply these approaches to the "noisy bars" problem.

3.4.4 Sparse Representation

Sparse Representation is an attempt to find a middle ground between Local and Distributed Representation. In particular, it looks to combine the non-linearity of Local Representation with Distributed Representation's advantages of being able to use more than one output at a time. However, the field is an extremely new one, and no algorithm has emerged that is clearly "best". In this section, I will discuss two promising examples of algorithms that produce sparse representations.

Before detailing these algorithms, it is useful to look at an example of what Sparse Representation is supposed to do, in comparison to Local and Distributed Representations. In this case, the "noisy bars" problem (as suggested in [13]) clearly shows the distinction between these three approaches. The data presented in this example is taken from [14].

The major reason for presenting this example is to show that sparse representation systems will generally find a representation that is similar to a local representation, except that it has the capability of having more than one output with a large value. Most output values will still be close to zero. Understanding the internals of this process is not necessary for understanding its usefulness.

The input in the "noisy bars" problem consists of 36 elements arranged in a 6 by 6 grid (larger grids are, of course, possible). There are two types of inputs possible: horizontal bars and vertical bars. A particular input will have either horizontal or vertical bars, but not both. Each of the 6 possible bars has an equal probability of appearing (generally around 0.3-0.4). Thus, possible inputs could be:

```
111111       000101       000000
000000       000101       000000
111111   or  000101   or  000000
111111       000101       000000
000000       000101       111111
000000       000101       000000
```

A significant amount of noise (random variation in the range of plus or minus 0.5) is also added to the inputs, resulting in input data of the following form:

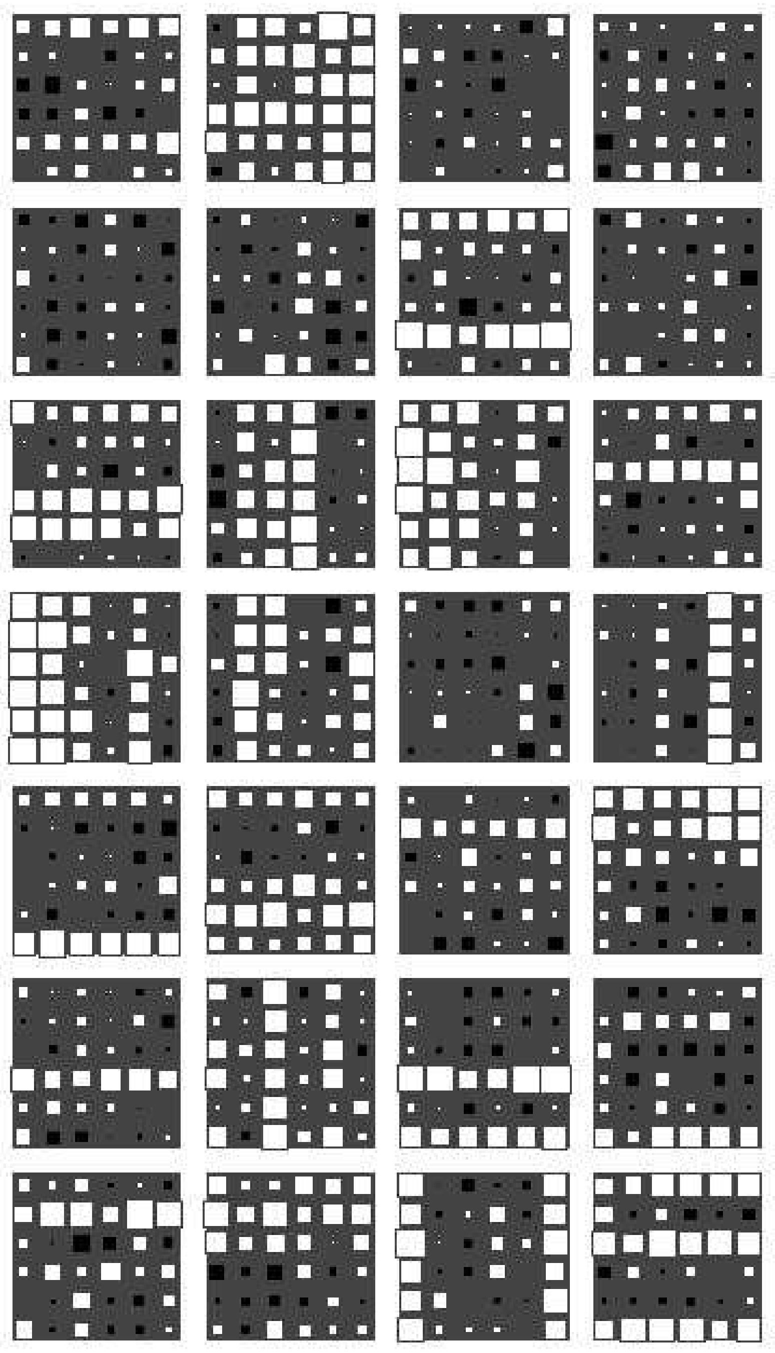

Figure 3.7: 28 different sample inputs for the noisy bars problem. The size of the square indicates the magnitude of the input, with white being positive and black being negative. (From [14])

We can now give this data to a Local, a Distributed, and a Sparse network and see what characteristic results each gives. For the sake of comparison, we shall keep the number of output nodes constant across the networks at a value of 24.

A Local Representation network (such as a basic classifier) is quite capable of dealing with the effects of noise. It will, generally, recognize the same pattern of bars under noisy conditions. However, since there are 126 different possible arrangements of the bars, it would need 126 output nodes to represent them all. As there are only 24 output nodes, what the system does is develop outputs that correspond most strongly to common combinations of bars, as in the following figure:

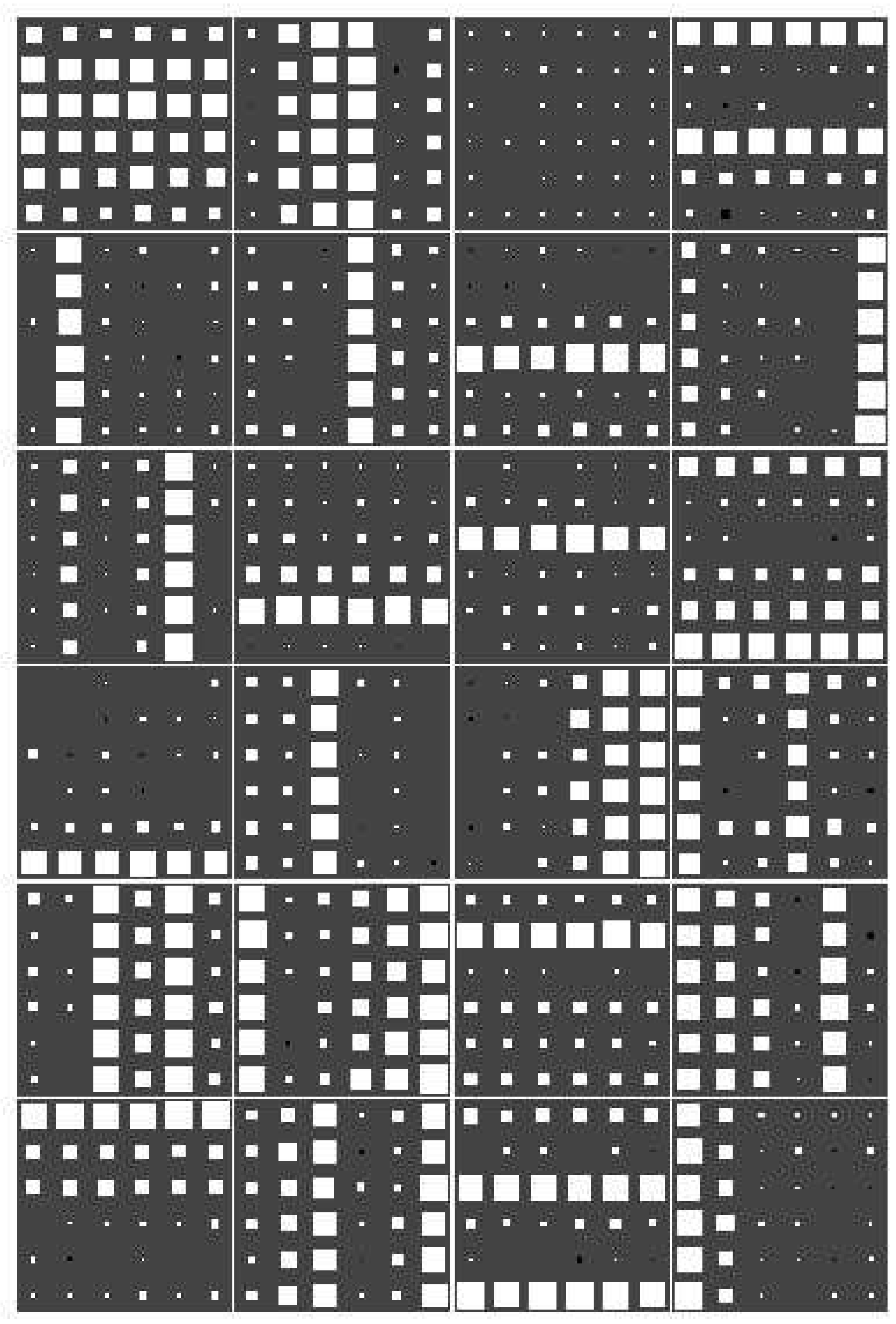

Figure 3.8: Results from a classifier system (local representation) on the noisy bars problem. Each of the 24 large blocks represents the input pattern that one particular output neuron responds most strongly to (i.e. what pattern that value in the output represents). The size of the small squares in the blocks indicates the magnitude of the input, with white being positive and black being negative. (From [14])

It can be seen that a number of the output nodes are responding to the presence of multiple bars. Thus, the presence or absence of any particular bar is not something that can be readily extracted from this representation.

The networks that produce distributed representations are more difficult to understand. In this particular example, they in fact produce much less useful results than the local representation systems. The following diagram shows how the individual output nodes correspond to strange mixes of positive and negative bars.

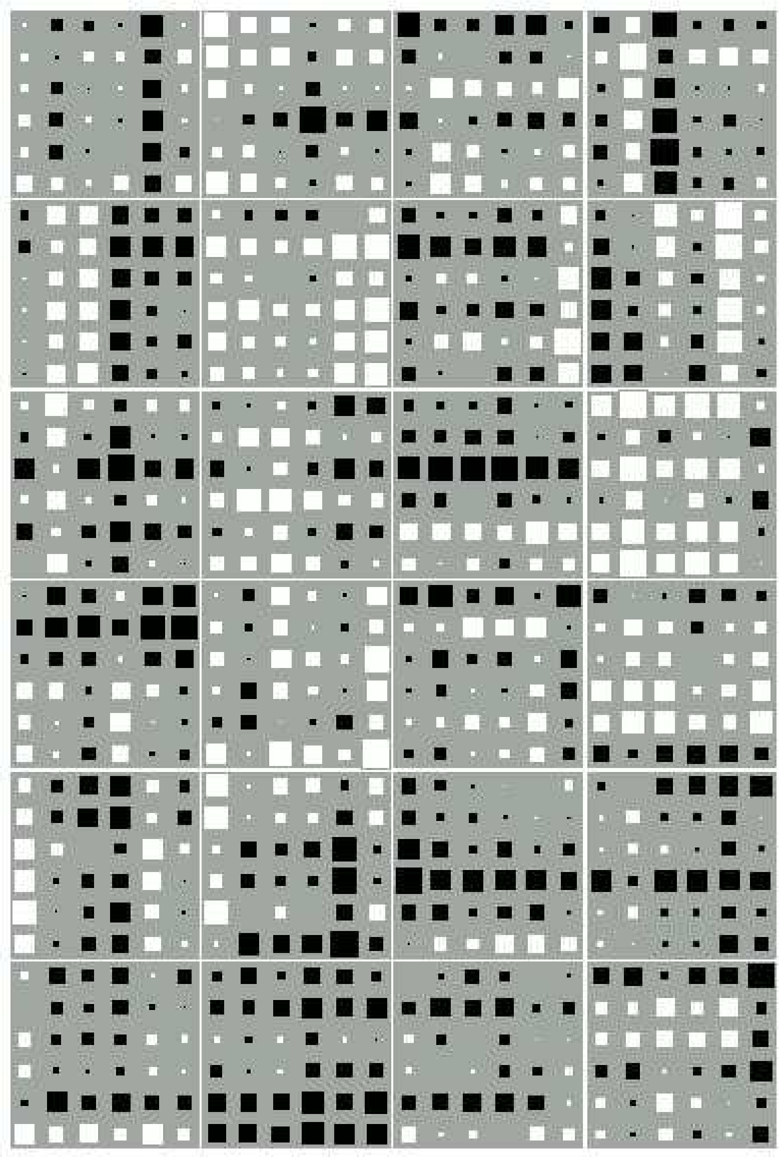

Figure 3.9: Results from factor analysis (distributed representation) on the noisy bars problem. Each of the 24 large blocks represents the input pattern that one particular output neuron responds most strongly to (i.e. what pattern that value in the output represents). The size of the small squares in the blocks indicates the magnitude of the input, with white being positive and black being negative. (From [14])

About the only thing that is clear from this example is that looking at each output neuron individually does not give us any sort of useful representation of the pattern at all.

This is clearly not what we want in a representation. The representation that would make the most sense (to us) would be one that uses 12 output nodes to represent the 12 possible bars. This is exactly what a sparse representation network would do. Both of the models (RGBN and REC) that will be discussed in more detail have been tested on this problem, and both produced exactly this result. If they have too many output nodes, then the extra output nodes are not used. 12 of the nodes, however, exactly correspond to the 12 possible bars.

Of course, the goal is not to form representations that are useful to us, but rather to form representations that are useful to whatever is using the representation. However, it does seem to be telling that these two very different more general techniques both generate representations that match our own expected representations.

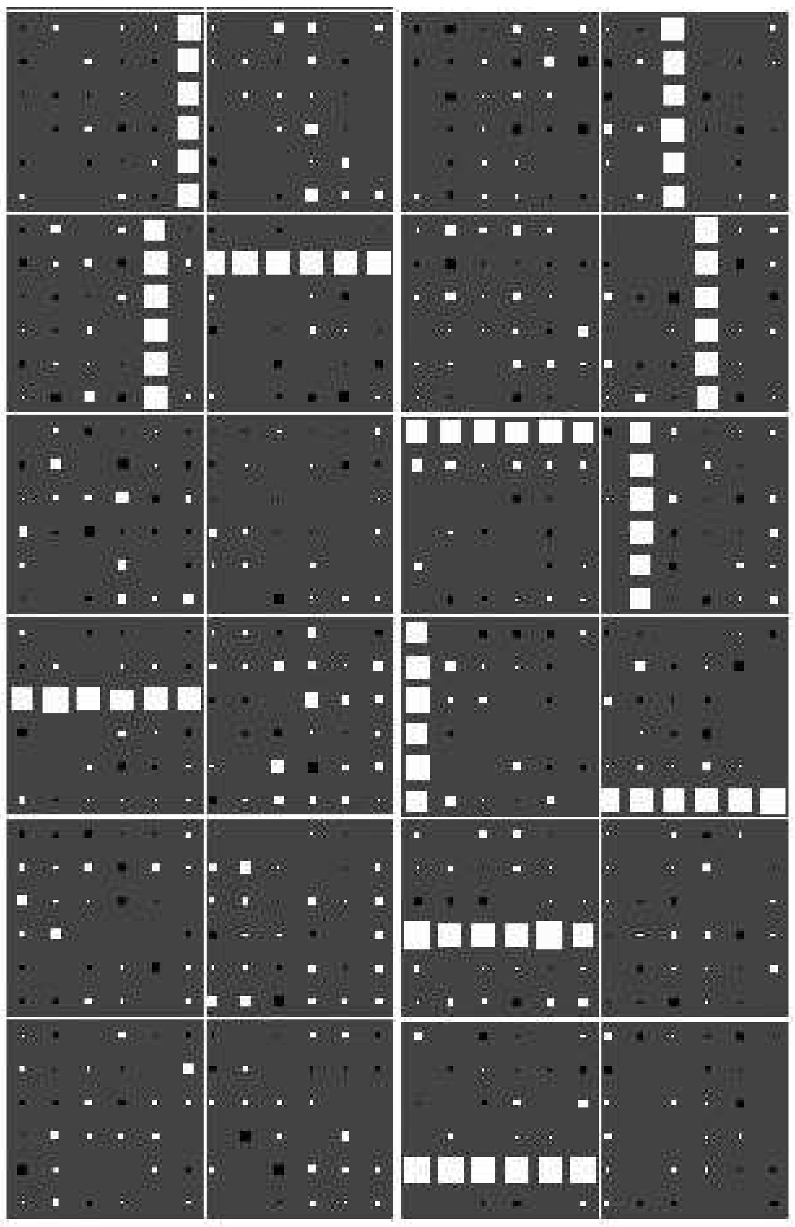

Figure 3.10: Results from RGBN (sparse representation) on the noisy bars problem. Results of the REC algorithm are similar. Each of the 24 large blocks represents the input pattern that one particular output neuron responds most strongly to (i.e. what pattern that value in the output represents). The size of the small squares in the blocks indicates the magnitude of the input, with white being positive and black being negative. (From [14])

We can clearly see that the sparse representation approach identifies the twelve different possible bars. The representation that it forms for the inputs are also highly efficient, as the network only uses twelve of the available 24 neurons to form its representation (the other neurons end up trying to model the noise). Importantly, it has picked out individual, separate aspects of the input. This means that a larger system which used this sort of representation could then make use of the fact that one particular neuron in this sparse representation signals the presence of, say, a horizontal bar at the top. None of the other representation systems would be capable of this.

The algorithms to achieve sparse representation are, in general, more complicated than those for local or distributed representations. One major reason for this has to do with the idea of *generative* weights. Generative weights are a set of backwards connections in the neural network that allow the network to act in reverse. For example, a local representation system running backwards would allow us to specify a particular classification by fixing the states of what are normally the output neurons, and the network would output (via the normal input neurons) an example (or prototype) of that class. A similar feat would be possible with the distributed representation networks. The important point is that this process is very simple with the classifier systems (local representation) and with principal component analysis (distributed representation). With the classifier system, only one output is active at a time, so we can determine the best example of an input that would create that output simply by looking at the forward connection weights (also known as recognition weights). With principal component analysis, we have the assumption that everything is linear, which means that we can look at each output individually, do a similar process as we did with the classifier system, and add everything together to get the final prototype.

However, sparse representation explicitly allows multiple inputs to interact in a non-linear manner. This causes what is known as the "explaining away" problem. Since most learning algorithms depend upon doing backwards processing, it causes the learning to be much more processor-intensive, and generally non-local. Indeed, a great part of the difficulty in developing networks that create sparse representations is to allow this process to happen in a neurologically plausible way. Neurons are inherently "local" devices, meaning that they do not have access to any information other than that of the neurons directly connected to them. Any large-scale algorithm that required neurons to use data from all other neurons in the network would not be a realistic model of a biological process.

To get an idea of the "explaining away" problem, consider a simple neural network with two inputs and two outputs. Assume all the inputs are connected to all of the outputs with a connection weight of 1. Now, if we are feeding forwards, and we input (0,1), we will get an output of (1,1). If we input (1,0), we will also get (1,1). When we want to work the network in the other direction, and we give a (1,1) to what are normally the output neurons, how can we determine what the input should be? If we set either neuron in the input to a 1, then that "explains" both of the values in the output. Note, of course, that this is an extreme example and has no clear solution, but it shows that in general, it is non-trivial to get a neural network to work in both directions.

However, in recent years, there have been some promising developments in this field. The following two sections will look at two algorithms in particular which have shown promise in developing these sparse representations.

The Rectified Gaussian Belief Network

The Rectified Gaussian Belief Network (RGBN) [14] is a neural network model which tries to learn non-linear distributed representations in a neurologically plausible manner. In practice, the representations that it does form are in fact sparse. It is roughly based on the Boltzmann machine, in that the generative weights (see section 3.4.4) form an important part of the model. However, the architecture is that of an acyclic graph (in other words, there are no loops), which greatly simplifies the mathematics required.

In order to introduce non-linearity into the system, the RGBN model limits each neuron to only outputting a positive value. This seemingly simple modification is the major difference between it and the linear distributed approach (see section 3.4.3), and both complicates the mathematics involved and allows the network to find patterns that more closely match our conception of patterns.

One important aspect of the RGBN is how it deals with the "explaining away" phenomenon (the non-linear interference between the generative and recognition weights). The model adds lateral connections between individual neurons in the same layer and uses these connections to coordinate the learning. Of course, it is infeasible to have connections between all of the input neurons in a large network (as that would be an exponentially large number of connections). However, if these lateral connections are only between neurons that are physically close to each other, then we are led to an interesting situation. This situation is that neurons that are close to one another will be ones which respond to correlated patterns, and neurons which are far away, and thus not directly connected, will generally respond to uncorrelated events. This is a possible explanation of the phenomenon of neural topographic maps, which is the general observation that neurons in the brain tend to be organized in a reasonably coherent manner, with neurons that respond to similar stimuli relatively near each other.

The Recurrent Error Correction Model

A completely different approach to this problem, but one which gives similar results, is the Recurrent Error Correction model [12]. Instead of looking at the problem from the perspective of "reducing the number of dimensions while keeping the maximum amount of information" (see section 3.4.3), REC takes the approach of reducing the predictability in the data.

Predictability is generally defined in a mathematical, information-theoretic manner, but for the purposes of this document, we can use a simpler, "intuitive" definition. Suppose we have a sensor which returns two bits of information; that is, it gives two pieces of information, each of which can either be a zero or a one. Now, let us suppose that whenever we use this sensor, the two bits of data are always the same. Whenever one is a zero, so is the other, and the same with a one. The second bit of data is entirely predictable based on the first bit. It would thus be more useful to replace these two bits of data with a single bit. Note that the same argument holds true if the two bits are always the opposite of each other: we could still replace them with a single bit.

This sort of process could be applied to, for instance, the data coming in from the eye. Unless we are looking at television static, the colour we see at one point is generally pretty much the same as the colour we see at the point right next to it. Why don't we, instead of looking at the colour at every single point, just simply identify the colour of a region, and identify the locations where the colour changes drastically (i.e. edges)? Better yet, what about a pre-processor that takes the point-by-point image received from our eyes and converts it into something like "table with three chairs"? Surely that would be a very useful format, and it certainly reduces the data down to something which makes it much less self-predictable. In other words, what we want is a network that takes the input data and outputs it in a form where it is not highly redundant.

Of course, it is not that simple. That sort of pre-processor is well beyond the technological capabilities of artificial vision researchers, and furthermore, such extreme measures would be very domain-specific (that is, there would have to be a very different system in place for vision and for hearing, for example). What we really want to do, it seems, is to arrange things so that all of the bits of information the sensor gives are completely independent of each other. Interestingly enough, this problem has been very thoroughly investigated and forms the foundation of Information Theory. The solution is, very simply, data compression. This is in widespread use in almost every computer when it is necessary to transmit a large amount of data from one place to another place in the shortest period of time: you don't want to waste time by transmitting data that's predictable based on data already transmitted.

However, it turns out that a pure data-compression approach is not the best approach, since it loses useful regularities in the data. Looking at it as a compression problem ignores the fact that we are going to actually need to use the data in its reduced state. It would be good if the data "000000000111" was actually similar to the data "000000000110". Compression techniques do not generally lead to situations like this: generally, if you change one bit of data in the compressed data, it completely changes the meaning of the entire message. Thus, the sort of "compression" we are after is compression that gives a useful compressed format. The point is thus not purely the compression, but rather the reduction in redundancy.

The REC model solves this problem in a rather non-intuitive manner. It makes explicit use of the fact that the systems we are interested in are, in reality, composed of neurons. This is important in that it takes a lot of energy for a neuron to fire. This would mean that it would be to the benefit of an organism for its neurons to fire as seldom as possible (i.e. the output from the system should be more like "000000110001" rather than "010110110101". By adding this aspect to the general approach of reducing predictability, REC ends up developing the sorts of sparse representations for which we are looking. We also note that the sorts of mappings it creates have neurophysiological significance; when REC is used to analyse an image, mappings result which very closely match the feature analysis functions found in the visual systems of mammals. This includes forming specific neurons for detecting lines in various orientations.

Comments

My major interest in Sparse Representation is that it strikes me as fitting very comfortably within the TNGS framework. The sorts of representations that are found would seem to be ones that would be correlated with other representations in a meaningful way. Sparse representation algorithms seem to offer a way of producing similar results to what TNGS produces with its randomized networks, but in a more directed (and thus more efficient) manner. Instead of producing purely random representations and hoping for the best, we may be able to guide these structures, through the aforementioned algorithms, to produce the sorts of representations that we recognize. Local and distributed representations do not seem to be the sort of representation upon which TNGS would function well.

Criticisms

As interesting as these approaches to automatically finding patterns in the input data may be, there are a few issues that should be considered. The first of these is the recognition that none of these representation models have any sort of memory. As commented upon earlier (see section 3.3.3), neural networks with recurrent connections (loops) can base their outputs on previous data, not just on the data currently presented to the network. The representation models discussed here have no capability of doing this, and there does not seem to be a feasible way to give them this capability. It is true that it is possible to specify the inputs so that the input to the network is actually just the values of one particular sensor over a specified amount of time, but that still will not lead to the richness of possible uses of memory available to a randomly formed network. Of course, it is true that very few randomly formed networks would give the sort of useful representations that these specifically designed models will give.

Another issue is that each of these models finds only one particular type of pattern. It is unlikely that there is only one type of pattern that would be useful: one of the great advantages of TNGS is that it can, in theory, find any sort of pattern. Since all of the models discussed here find very different patterns, it is conceivable that all of them would be useful. We should not be looking at these models and asking “Which one is best?”. Instead, it may be more fruitful to use all of them at the same time.

The real question is, of course, “Is it useful?”. Do these networks find patterns that are at all useful to artificial agents interacting in a complex environment? None of the experiments using any of these models have used the sort of data that would be encountered by an agent’s sensors. The input data has been still pictures or made-up examples. To really see which of these models would be useful to an interacting agent, we would need to actually implement them in an actual robot (or a simulated one in a reasonably complex simulated environment) and see what patterns it is able to find. This idea will be the focus of the chapter on Artificial Life (see section 4).

3.5 From Subsymbols to Symbols

It is now perhaps relevant to make a few high-level comments about what the discussion so far might mean for cognition in general. In particular, the methods described seem to point to how a system might form representations (or concepts) about its environment. In particular, the concepts (patterns) that the system (organic or artificial) forms are highly related to predictability and regularity in the environment. Since the mappings develop over time as the agent interacts with its environment, the regularities and patterns that it finds will be entirely dependent on the sorts of experiences the agent has.

Firstly, this idea forms an interesting correlation with the result from developmental psychology that creatures who are not exposed to various different stimuli during the initial stages of their life do not develop the capability to detect that stimuli. Two standard examples of this are kittens raised in environments with no horizontal lines, and children raised to speak a language where certain phonemes are not used. In the first case, later neurological examinations of the kittens show that the horizontal feature detectors which generally exist in the mammalian brain do not form. In the second case, experimentation shows that, for instance, people whose first language is Mandarin have great difficulty distinguishing the “r” and “l” sounds familiar to native English speakers.

Secondly, this hypothesis about how concepts are formed directly predicts that people with different backgrounds will have difficulty communicating. Different backgrounds lead to different concept formation, and so even if people are using the same word to describe something, the environmental patterns that they associate with that word may be very different. This also means that there is no one “right” definition of a word (except, perhaps, for words and concepts defined in abstract mathematical terms which do not have experiential counterparts). However, communication is certainly possible to some degree, since there is significant overlap between the patterns that people in slightly different environments develop.

Chapter 4

Learning in Artificial Life

4.1 What is Artificial Life?

A brief introduction to Artificial Life was given at the beginning of this paper (see section 1.1). At this point, I would like to revisit the concept and look at it in the light of the information presented thus far.

Artificial Life has one very important distinction between it and other forms of Cognitive Science research. In particular, there is generally a complex and long-lasting environment with which the organism interacts. This is in contrast to the sort of situation discussed in the section on pattern discovery in neural networks (see section 3.4). In that case, the experimenter would simply present the system with a long series of inputs, and the system would learn to find patterns. In Artificial Life situations, the inputs to the creature are determined by its current location in the world. Furthermore, its location in the world (and other features in its environment) can be changed by the creature's actions. This means that the creature can, to some degree, control its own future input. It is this capability which opens the door to the types of powerful sensorimotor interactions discussed earlier (see section 2.4.2). This gives the creature a completely new range of behavioural possibilities, and necessitates a different way of analysing and understanding their behaviour [25].

This is not to say that the sorts of approaches used in non-Artificial Life research cannot be applied here. Patterns in input data can still be found – the difference is that they may be a different sort of pattern. For example, the input data to the creature will generally change only slightly over time, as the creature's situation in the environment is fairly similar from one instant to the next. However, we must also realize that the creature must actually *do* something in Artificial Life situations. The neural networks discussed previously merely had to find patterns and form representations. Artificial Life means that outputs are required on a continual basis. The creatures must move around and interact with their environment, not just passively observe it.

There are a number of other ways in which research in Artificial Life is unique. The majority of the work is done in "distributed" intelligence, where not only do we have behaviour being the result of a number of simple interacting rules, but these rules are implemented within multiple interacting creatures. These creatures (much like ants, flocks of birds, or schools of fish) perform group behaviour while acting individually, without any centralized control. Another unique aspect of Artificial Life is its interest in "life as it could be". Rather than dealing with the specifics of life as we know it, we can explore other possibilities that are not bound by the physical constraints of our world. However, for the purposes of this thesis, I will be dealing specifically with the interactive nature of Artificial Life.

4.2 Simulation versus Reality

Before delving further into Artificial Life, it is important to take a close look at one of the aspects of it that has caused a great division among researchers in the field. This is the question of using actual physical robots or using computer simulations of virtual robots. In both cases there is an agent interacting inside an environment, but the important differences are (a) the complexity of the environment, and (b) the amount of control the researcher has over the environment.

Artificial Life has its roots in both simulation and physical embodiment. The initial work which spawned the field included examples of both approaches. Conway's Game of Life [11], and Reynolds' "boids" [27] are prime examples of early work which used simulation to show that a very simple set of rules can create "life-like" behaviour. There is also the striking example of Grey Walter's light-seeking "tortoises" [44] which are arguably the first physical human construction which exhibited "life-like" behaviour. However, it soon became clear that the difficulties of working with physical robots made simulation a much more attractive option. With simulation, you don't have to worry about motors breaking or vacuum tubes burning out or indeed the rather difficult process of constructing a robot in the first place. Instead of a complex drive-train and steering system, with batteries to maintain and cables which tangle, you can simply specify that one of the actions the robot can do is "move forward" and another is "turn". Sensors are also greatly simplified, as you don't have to worry about variations in light levels and reflectivity, which make vision and infra-red sensors notoriously difficult. In fact, you can even have sensors that indicate if the object in front of the robot is "food" or a "wall" or a "predator". These sorts of things are certainly not easily done in a real robot.

However, these advantages of simulation may also be severe disadvantages. In particular, many of the issues that simulation allows you to avoid *are the very problems that need to be solved.* How does a living being identify something as "food"? How does it convert complex muscle coordination into "moving forward"? How do living creatures deal with the uncertainty and complexity of the natural environment? By trivializing these questions, researchers working in simulation can miss a very significant piece of the puzzle of understanding how life works.

This is not, however, to say that all simulation is bad. First of all, the question of understanding how life works is so large that insights can certainly come from looking at only one part of the whole issue. Secondly, all of these objections to simulation are not against something inherent in the concept of simulation itself. After all, it is certainly possible to simulate complex musculatures for walking, and one can also avoid using unfeasible sensor systems. We can introduce random variation (noise) into the simulation as well, and make things unpredictable. Our ability to create powerful and complex simulations is increasing faster than our abilities to create complex, robust robots, and so it perhaps makes sense to follow this route, rather than abandoning simulation altogether.

However, I find a third reason an even more compelling argument for the use of simulation. This reason is simply that, in a simulation, *you can speed up time*. If we are looking for artificial systems that can learn through experience, then there must be the time for them to gain that experience. If that time is done in a physical robot, then who knows how long it may take to learn required skills. Consider how long it takes a human baby, which is perhaps the most capable learning machine known, to develop basic motor skills. Would it not be much better if a robot could learn these skills in a simulated environment at a greatly increased speed? In fact, if we are going to be studying "learning machines", and trying out various different learning algorithms, then it is vital that we can try them out at high speed. We do not want to evaluate these algorithms by running each one on a robot for a full year and seeing which ones (if any) learn to walk. This would take ridiculous amounts of time and make it infeasible to try out lots of variations on a particular algorithm. Artificial Life is an experimental science, and simulation is necessary for these experiments to take place. For more discussion of these issues, see [21].

We are left, then, with the problem of how to do realistic simulations. What we want is a simulator that is good enough that we can take the agent, once it has learned to do things in the simulator, and put it into a real, embodied robot and it should be capable of performing the same tasks. It is, of course, likely that once in the real world, it may need to do a bit of fine-tuning of its learning (one wheel may have slightly more friction than it did in the simulation, for example), but overall it should work fairly well. Unfortunately, this has generally not turned out to be the case. Most attempts at this have been disastrous, which is the general argument for why we should not be using simulation in the first place.

There have been some interesting approaches to solving this problem. Most notable is Jakobi's "Envelope of Noise" hypothesis in his "minimal simulations" [16] [17]. He has had remarkable success in having agents transfer perfectly from simulation to reality in a number of different domains. His approach is to drown the system in large amounts of noise (random variations in the sensory system and in the environment itself). A very few things are decided upon as being important values that the robot can rely on (and these are generally things that good sensors do physically exist for, or basic fundamentals of the real-world physics), and everything else has more noise than the randomness in the real world could ever provide. Importantly, this noise is of a very particular form, in that the amount of noise is, itself, a random amount. This is to stop the agent from actually learning to *use* the noise (i.e. the agent expects its sensors to be particularly noisy, and then its learned skills no longer work when that noise is removed). It is also interesting to note that the use of noise means that the simulation itself does not need to be at all accurate. We don't need to simulate exact physical collisions of solid objects in three dimensions if the noise is going to make all of that extra calculation unnoticed. The resulting high-speed, bare-bones simulations seem to offer hope for useful simulations.

Before leaving this topic, it is also relevant to note that even people who do work with physical robots are guilty of the same simplifications that simulators make. In particular, almost all experimentation with physical robots has taken place in "toy worlds". The environment that robots deal with is not the complete complex "real-world" that animals live in. They are generally constrained to flat, hard floors, or inside mazes, generally with all-white walls to simplify vision. These simplifications, while much less extreme than those encountered in most simulations, are nevertheless the same sort of simplification that may cause eventual problems. However, as stated before, the field of Artificial Life is still in its early stages, and different amounts of simplification are right for different sorts of research.

4.3 Distributed Adaptive Control

We will now examine a learning method that was developed explicitly to be a model of classical conditioning within an embodied, artificial life context. This method, known as Distributed Adaptive Control (or DAC) was developed at the University of Zurich Artificial Intelligence Lab, and has been the subject of a number of experiments and publications. The details of the algorithm are given in [41]. In this section I will describe the method and relate it to the work of other researchers that has already been discussed.

The roots of DAC start in a paper by Verschure and Coolen [39], where they develop two connectionist (i.e. neural network) models of classical conditioning (see section 3.2). These models start with a system where there is an unconditioned stimulus that causes an unconditioned response. Over time, if a new stimulus is also given at the same time as the unconditioned stimulus, then the system will learn to associate the new stimulus with that same response. Furthermore, their models have four other special properties.

Firstly, their models do not use a local representation. Previous computer models of classical conditioning, such as the Rescorla and Wagner model [26], used local representation, and this causes all the problems discussed earlier (see section 3.4.2). In particular, it means that there must be a separate neural input for each and every possible stimulus or combination of stimuli that the organism might encounter. In other words, the organism must have a pre-specified neural connection for each and every thing that it might want to learn during its lifetime. There is no sense of "similarity" between different sensory events: two events are either both of the exact same type (and are treated identically), or they are of completely different types. This is an unfeasible situation, as it causes a combinatorial explosion of connections, and does not allow for the sort of generalization that living creatures require. Instead of taking this route, the DAC models make use of any arbitrary type of representation. This can be everything from a local representation to a fully distributed representation. The system simply accepts whatever patterned inputs are given to it, be they local (only one value of the input set to 1, and the others 0), distributed (arbitrary patterns of 1's and 0's), or sparse (mostly 0's with a few 1's). What this means is that these models can handle an aspect of classical conditioning which is not generally studied: the ability to discover new stimuli.

Secondly, their models are neurologically plausible. The individual components that make up the model are standard artificial neurons, and the learning rules used by the system are ones that would be feasible to find in the brain. Learning is based on the strengths of connections between neurons, and the rule that specifies how a connection weight should change does not require information about the state of the entire system.

Thirdly, their models exhibit blocking. This is a phenomenon found in natural studies of classical conditioning, and deals with situations with multiple new stimuli being learned. In these cases, experiments show that the learned association strengths between the new stimuli and the unconditioned stimuli are not independent of each other. Instead, the one new stimulus with the most predictive power (i.e. the one that is experienced most often with the unconditioned stimulus) keeps its strong association with the unconditioned stimulus, and the other stimuli get a lower association than they would normally. This is similar to the "explaining away" phenomenon covered in the discussion on neural networks (see section 3.4.4). It is also related to the process of extinction, which defines how quickly (or slowly) an organism will lose an association that it has learned, if the environment changes such that the new stimulus is no longer a good predictor.

Fourthly, their models allow for second order conditioning. This occurs when a system that has already learned a new stimulus to associate with a response (based on an original unconditioned stimulus-response pair), and then learns to associate another new stimulus based on it occurring at the same time as the previously learned new stimulus. In the classic Pavlov's dog example, this would happen if, once the dog had learned to associate the sound of a bell with the arrival of food, we then started blowing a puff of air at the dog whenever we rang the bell. The dog would then learn to associate the puff of wind with the arrival of food, and would soon be salivating in response to the puff of air on its own.

The two models that Verschure and Coolen present are very similar, with one being an extension of the other. The models are based on very simple single-layer groups of neurons that are fully interconnected (referred to as fields). Initially, the interconnection weights of the neurons are configured so that if we present the system with an unconditioned stimulus (i.e. set the activations of all of the neurons to a particular, pre-specified pattern) and then leave the system alone, the pattern of activation will change to the unconditioned response (i.e. the neurons will excite each other until they settle into a different pre-specified pattern). We can then present the system with a new stimulus (i.e. a new pattern of activation), followed by the original unconditioned stimulus. Over time, it will learn to associate the new stimulus with the unconditioned stimulus, and thus elicit the response (i.e. we can set the neurons into the new pattern, and the interconnections will cause the neuronal pattern to change to the unconditioned stimulus pattern, which will then change to the unconditioned response pattern). The second model extends this idea to multiple sensory modalities, with multiple different groups of neurons corresponding to different types of inputs.

The learning rule that is used is basically the standard Hebbian learning rule, where the connection between neurons is increased if both are active. However, this had to be modified slightly in order to produce the blocking behaviour desired. This modification takes the form of "competition" between the connections. The change in the weighting strength of a connection ends up being weighted by the current strength of the connection itself. Note that this modified form of Hebbian learning is still a local learning rule (the "competition" is between connections of one neuron, not the whole system), and so the system is still neurologically plausible.

The actual DAC learning algorithm is not exactly either of these two models, as various approximations and modifications were made to transfer the system from its theoretical format to one that is implementable in a physical robot. A complete description of the actual DAC architecture and its more recent extensions, with particular focus on its connection to classical conditioning, can be found in [43] and [38]. The DAC system does, however, seem to inherit the major properties of the theoretical models.

4.3.1 Implementation

This general approach forms the basis of the Distributed Adaptive Control algorithm. DAC takes this method of learning and places it inside a mobile robot (either in simulation or in reality), and the robot will learn to move around and avoid obstacles. In order to understand how it does this, we first need to look at the physical structure of the robot and its possible interactions with its environment. The following information refers to the first example robot with which the Zurich group worked. Since then, DAC has been applied (and extended) to a number of different robots and tasks.

The basic robot used has three different types of sensors. It has 37 collision detectors arranged around the front half of its body. These are triggered if the robot hits an obstacle with that part of its body. There are two "ears" which detect the distance to a target, one on each side of the robot. There are 37 range-finders arranged around the front half of the body. These are able to detect how far away a solid object is from the robot.

Initially, the robot has a set of built-in stimulus-response pairs. These form the unconditioned stimuli and responses. In particular, if the robot hits an obstacle on its left side, it will back up and turn right. If it hits something on its right side, it will back up and turn left. If the “ears” detect a target on its left (or right) it will turn to the left (or right). If none of these stimuli are occurring, then the robot will go forward. Note that the robot’s basic repertoire of behaviours does not use the range-finders at all. Instead, the DAC algorithm applies its learning algorithm to associate the range-finder stimulus with the collision and target-finding unconditioned stimuli. This learning is done in real-time as the robot explores its environment. Initially, the robot will run into walls, since it has not yet learned to use the range-finders. However, the robot will very quickly associate a range-finder value indicating “near a wall” with a collision, and so it will learn to turn away before the collision takes place. It is thus able to use the range-finder as a predictor of the collision stimulus.

In this particular case, it is relatively straight-forward to envisage how this learning takes place. As the robot wanders through its environment using its set of initial movement rules, it will soon collide with an object. During the moment of collision, and, indeed, during the time just before the collision, the range sensors will be indicating a particular pattern (high values for those sensors pointing directly in front of the robot). After a few similar collisions, the DAC learning algorithm will have learned to associate the high range-finder readings with a collision. The input of a collision is the initial unconditioned stimulus, and the act of backing away from the object is the initial unconditioned response. The high range-finder values are the new stimulus, which the algorithm causes to be associated with a collision, and thus backing away. Thus, in the future, when the robot encounters a similar situation, the range-finder values will be of a certain pattern, which will cause the “collision” stimulus to be formed (even if the robot has not actually collided with the object yet), which will cause the robot to turn away from the object. This leads to the robot being able to navigate around its environment without colliding.

The simplest experiment with this control structure is to place the robot in a walled environment with many obstacles and count how many targets it finds (a new target would appear on the other side of the room once the robot found the current target), and how many times it collides with an obstacle. According to [40], a typical run of 5000 time-steps would consist of finding 32 targets while colliding 52 times. Finding targets would happen at a very consistent rate, while collisions would stop happening very early into the experiment.

Various other experiments with this architecture have given interesting and promising results. For example, in one experiment, the robot was placed in an environment where targets were often found near walls. In this case, the robot learned a general wall-following behaviour [42]. A more complex avenue of research was followed in trying to add the ability to learn a sequence of actions [43]. The task in this experiment started with the robot moving around in its environment as before,

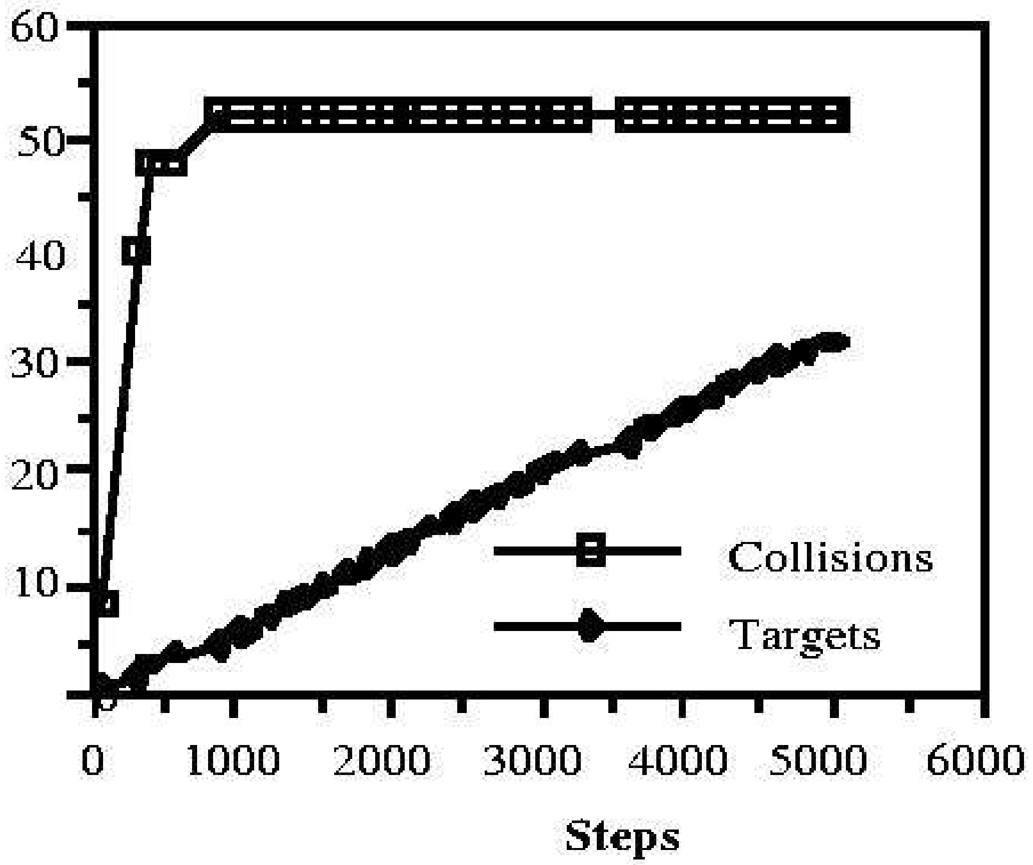

Figure 4.1: Copied from [40], figure 3. Collisions and target finding events for a typical run of the basic DAC architecture.

finding targets using its target sensor and avoiding obstacles using its learned responses. In this case, the targets were placed at two specific locations, so the robot would travel to one location, then to the other, and repeat the cycle over and over. Then, the target sensor was removed from the robot, so it could no longer sense the target from a distance. However, the robot was still able to robustly find the targets by simply following its learned temporal associations. Because it had been following the same route many times, it had built up associations along the path it was following. So, for instance, it might be associating turning left with a particular junction in its path. Once the target sensor was removed, it would still turn left at the junction, simply because it had done so in the past. This seems to be a step towards the phenomenon found in natural creatures of associating an action with a place, which was mentioned earlier (see section 3.2). Originally, this addition to the architecture (called DAC3) was not biologically plausible, as it used an internal database of previous states for a memory. However, a new implementation was able to achieve the same results in a biologically plausible manner [38].

4.3.2 Comments

Distributed Adaptive Control is important for our consideration, as it seems to be a successful model of classical conditioning being used to control a mobile robot interacting with its environment. It shows us how this idea of "forming associations" can be instantiated, and gives us an idea as to how a robot (or, indeed, how a living creature) could use associations to act. It is also very simple, and is expandable to handle multiple sensory modalities. Furthermore, the four properties of the model listed earlier (see section 4.3): non-locality, plausibility, exhibiting blocking, and second-order conditioning, are all vital to any model which tries to explain (and use) the sort of associative learning found in living creatures.

4.3.3 Criticisms

There is one minor criticism that should be brought up here. It is something that has been commented on before, in the section on the Theory of Neuronal Group Selection (see section 3.3.2). In particular, it was noted that we may not always want to form associations between the raw sensory inputs to our system. DAC as it stands always forms associations using the data coming into the sensors, without any sort of preprocessing being involved. This means that the patterns with which it forms associations cannot be as complicated as those that TNGS may use. For example, DAC could not make use of a pattern that appeared to the sensors over time.

This indicates how DAC fits into the ideas discussed earlier in this paper. In particular, DAC can be seen as a specific implementation of the association behaviour inside TNGS. Much as unsupervised sparse representational networks could be used instead of the standard randomly formed "pre-processors" for a TNGS system (see section 3.4.4), DAC seems to be a possible alternative implementation for the internal functioning of TNGS. With basic TNGS, we form associations between the outputs of a large number of randomly formed pre-processed views of the sensory data. One advantage that DAC would have over the basic TNGS approach is that it would take into account the phenomenon of blocking (see section 4.3). Furthermore, the exact algorithm used in TNGS is left generally undefined. In any case, the philosophy and approach taken by DAC and TNGS researchers is clearly similar and compatible.

The major problem with DAC at the moment is that the properties of the learning process that it undergoes have not been strongly established. The experiments performed by the Zurich group have shown some classical-conditioning-like results, but this has been done purely looking at the associative aspect of classical conditioning. Their work has not looked at how well the system un-learns associations (extinction), nor has it explicitly looked at the generalization capabilities of the system. Another key part of classical conditioning is the ability to discriminate, a process whereby two stimuli are initially found by the organism to be similar, but after a period of training it can learn to make fine distinctions. It is the goal of the experimental part of this paper to do these sorts of experiments on DAC, in order to determine if it really is a feasible candidate for being the core of a more complex learning system.

4.3.4 DAC and Operant Conditioning

So far, we have focussed on Distributed Adaptive Control as a model of classical conditioning. Indeed, the papers published regarding DAC are all in classical conditioning terms. There is thus an open question as to whether this approach can be successful as a model of operant conditioning as well. Operant and classical conditioning are learning methods which share the basic property of forming associations between patterns, which is the main algorithm behind DAC. In this section, I which to discuss the possibility of DAC serving as the basis for both types of conditioning.

Put simply, operant conditioning is a process whereby a living creature learns to respond to its environment in order to be more likely to experience "positive" stimuli, and less likely to experience "negative" stimuli. There is a slightly circular definition here, as positive stimuli are defined to be those things that the agent tries to have more of, and the opposite for negative stimuli. However, positive stimuli tend to be fairly common among living creatures: food, warmth, affection, and other necessities. Negative stimuli tend to be detrimental to the creature's livelihood, such as electric shocks or illness. The impressive aspect of operant conditioning is that the actions that the creature can learn to do to achieve the positive stimulus can be very complex and nothing like the actions the creature would normally exhibit. For examples of this, the reader is referred to any circus or animal show.

Before going any further, it is important to not be confused by a point of terminology. In many of the DAC papers, the two hard-wired reflexes of backing-and-turning-after-a-collision and turning-to-face-the-target are referred to as the "negative unconditioned stimulus" and the "positive unconditioned stimulus". This choice of terminology, however, does not have any relation to the "negative" and "positive" reinforcement referred to in operant conditioning. The two basic DAC reflexes are simply built-in reactions, and calling them "negative" and "positive" is merely the judgement of the observer. DAC treats the two reflexes equally, and does not try to achieve the positive while avoiding the negative. To perform true operant conditioning, the creature must be able to modify its behaviour so that an action in a given situation which elicits a negative reinforcement signal should have its associations decreased, while one that elicits positive reinforcement should be increased.

There are two major differences between operant and classical conditioning that are generally cited. Firstly, the operant conditioning results in new behaviour, while classical conditioning merely causes an old behaviour to happen in a new situation. Secondly, operant conditioning is active, rather than passive, in that the creature initiates the action, rather than the experimenter. I would argue that neither of these distinctions is useful from an implementation perspective.

In the first case, we have the claim that operant conditioning produces new behaviour. My problem with this statement is that the term "new behaviour" has no objective meaning. Whether the behaviour of a creature is new is completely subjective. After all, a creature has a finite set of muscles or effectors that it can use, so at the low level of muscle control, there can clearly be nothing "new". For example, consider the learning that the basic DAC creature does when it learns to avoid walls. Its initial behaviour is to collide with an object and then turn, and after a while it learns to turn to avoid the walls. Is this a new behaviour? It certainly looks that way to an outside observer. However, at the low level, there is nothing new happening. What is new is how the motor actions fit into the sensorimotor loop. We can thus see that classical conditioning is capable of producing new behaviour, and the term "new behaviour" isn't particularly easy to define. In fact, I would recommend avoiding the term altogether.

The second case deals with the passive/active distinction. I would again argue that this is not a useful way of looking at the distinction between classical and operant conditioning. A learning agent is always embodied in a world: it is always receiving inputs and creating outputs simultaneously. In the so-called "active" operant conditioning, the agent is still stimulated by the environment to do something: it needs to be in a situation where it is appropriate to perform its action. A dog operantly conditioned to perform tricks to receive a reward will only perform the trick if there is someone there to give it the reward.

However, the active/passive aspect does bring up an important difference between the two. In operant conditioning, we reward a creature's behaviour *after* the action. Let us look at what this means within the context of the DAC architecture.

In the DAC system we have examined so far, we can present the system with a new stimulus, and it will then generate the original stimulus for itself, causing the hard-wired response. So, we present the creature with a high range-finder reading, it associates that with a collision, and thus it turns. Now, in an operant conditioning situation, the agent is going to have some environmental stimulus (the original situation the agent is in), it will perform some action, and then we will give it a reward stimulus. Note that the only real difference between this description of the events and that of classical conditioning is the order of events. A second seeming difference would be that instead of associating a stimulus with another stimulus, we are associating a stimulus with a response. Conveniently, this second difference is easily accommodated within DAC, since the things that it forms associations between are merely seen as data patterns: the DAC algorithm should deal with response patterns as well as stimuli patterns.

DAC would, however, need to be able to deal with the modified order of events. In particular, instead of simply forming associations between sensory inputs all the time, it needs to also be able to form associations based on whether that association resulted in a later "good" stimulus. (We can look at this "good" stimulus as being similar to that of an electrode implanted into the hypothalamus.) Also, this could be extended to "un-form" associations in response to a "bad" stimulus. Both of these are feasible within the DAC architecture, however, they would seem to require a form of memory to handle the temporal aspect. Fortunately, this should be a fairly short-term memory, as operant conditioning tends to work best in the 0-10 second interval range. A simple implementation of this approach would be possible, by storing the sensory and motor data for the past few seconds. However, this naive approach would not be neurologically plausible. Other possibilities do exist, such as using a more complex model of a neuron which changes its characteristics based on its firing pattern (i.e. the neuron has a simple memory of its past activity). This could be similar to the more complex neural models which keep track of the depletion of various neurotransmitters (for example, [36]).

4.4 Other Architectures of Note

Distributed Adaptive Control is one of the few algorithms that focuses on learning during an individual's lifetime in a neurologically plausible manner. However, there are a number of other ongoing research projects which also examine conditioning in embodied agents, although through very different means.

For example, Sutton's Dyna architecture [32] takes an approach based on GOFAI-style planning and maximizing expected outcomes. The agent builds and maintains a model of the world and tries out different possible actions in its model to discover the best current action. However, the system seems to have difficulty dealing with real-world problems, such as having multiple goals.

Also of note is the ELDEN architecture [48], which has been successfully run in embodied robots. It consists of three interacting learning methods, each covering a particular task. The first handles transient changes, which do not involve modifying the robot's internal world model. This would include people walking into the robot's path, causing it to stop and wait or go around. This is handled in a neural manner, similar to the standard Braitenberg vehicle approach. The second deals with constructing and maintaining a model of the world by building a cartesian spatial map of visited locations, based on its range-finder readings. The third deals with correcting for wheel slippage (the difference between the robot's desired motion and its actual motion due to real-world physics) by comparing current sensor readings to what the internal world map predicts. This model, however, requires that each learning method be hand-tuned and specific to a particular task.

The same researchers who developed ELDEN have also worked in purely neural models of learning. In [47], they use artificial evolution to develop neural networks which exhibit sequential behaviour. They use the genetic algorithm as a way of producing operant conditioning results: networks which perform well are kept to evolve further, while those which do not perform well are eliminated. By running this process over many generations, they are able to produce more complicated behaviours. However, this is done in a tightly supervised manner without a complex environment. [8] also uses a genetic approach to "shape" a system's behaviour, although in this case the evolutionary process creates production system rules rather than neural networks.

Another neural learning approach can be found in [35]. In this case, particular focus is made on forming internal representations through experience. It uses genetic algorithms and lifetime-learning techniques to accomplish this in a task which directly involves handling a dynamic environment. However, as it currently stands, the system does not handle noise in the environment well, and its implementation of memory is tuned for the particular task.

Chapter 5

Implementation

We now turn to the implementation phase of this thesis. The preceding text has discussed various issues surrounding learning in artificial life, and some of the approaches that have been taken. It is now time to put this information into practice.

The goal of this implementation is two-fold. Firstly, I wish to develop a number of tests which can evaluate how close a particular situated associative learning algorithm comes to exhibiting the complex characteristics of classical conditioning. Secondly, I wish to run these tests on the DAC learning algorithm. Through this process, we can identify those aspects of classical conditioning for which we do have a good model, and those aspects for which more research is needed.

Because the test suite will, to some degree, be constrained by the sort of implementation chosen for DAC, this section starts first with a look at the simulation engine itself, followed by a series of experiments which establish that it is actually a true implementation of DAC. Once this is done, the following section will describe the various conditioning experiments and their results.

5.1 Simulation Design

The simulation system used in this thesis is not one which, to my knowledge, has ever previously been used for cognitive science research. However, it is one that I have used for various purposes in the past, and which I find to be remarkably stable and suitable for this sort of work. In this section, the major features of the basic engine itself will be discussed, as well as the specifics of my implementation of DAC within this environment.

This section is targeted at an audience of those somewhat familiar with the principles and language of object oriented programming. It is also somewhat separate from the rest of this thesis, in that fully understanding this section is not a requirement for understanding the results of the experiments that will be discussed in later sections. However, if you are interested in simulation systems, I hope you will find the description given herein an interesting alternative to the sort of simulation environments commonly used in this field.

5.1.1 The Quake 2 Engine

All of the simulation work done for this thesis was performed using the Quake 2 engine [15]. Quake 2 is a highly popular commercial game developed by id Software and released in 1997. Like the rest of the company's games (Wolfenstein 3D, Doom, Doom 2, Quake, and more recently, Quake 3: Arena), the game focuses on high-speed graphics in a three-dimensional world, with significant interaction between the user and the objects in the world. However, the most important aspect of their software for the purposes of this thesis is the company's commitment to making their games user-modifiable.

With their first few games, this modifiability was in terms of allowing users to design their own levels for the game, using custom graphics and world geometry. However, starting with the Quake series, the company has exposed the programming interface to the game logic itself. That is, they have allowed users to completely change how the game itself works, not just what it looks like. This has led to an explosion of new games developed by amateur programmers who have adapted the Quake engine to drive a large number of new games which bear little to no relation to the original Quake series.

However, there seem to be very few people who are trying to use the engine for purposes other than games. This is unfortunate, since the Quake 2 engine offers the researcher a stable and fast three-dimensional environment, as well as a legion of interested programmers who are quite willing to answer questions and to help solve any problems that may be encountered.

One possible reason for the software's lack of use among non-gaming researchers is the fact that Quake 2 uses the C language for user modification. While C is a fast language, it generally causes aggravation in the debugging stages of software development, since the majority of programming errors cause the computer to crash (usually) without indicating exactly what the error actually was. However, this problem is addressed by the Q2Java project [23]. This project, run by an informal group of programmers, has developed an interface from Java to the Quake 2 engine. This brings all the benefits of Java into Quake 2, allowing detailed debugging messages and the use of the large (and growing) library of Java classes available online. Furthermore, the Q2Java project has developed a much cleaner programming view of Quake 2, in particular by reorganizing the system to separate the gaming aspects of the engine from the pure simulation aspects.

The following list gives the major features of the Quake 2 system.

- a fully three-dimensional environment made up of polygons of arbitrary orientation
- a large selection of modelling tools for designing the environment and placing objects within it
- movable world geometry (i.e. sliding doors, rotating towers, moving platforms)

- animated, fully programmable objects within the world
- collision detection between objects within the world and between the objects and the world itself
- networking support, allowing other computers to connect to the computer running the simulation and observe or affect what is happening
- separation of the simulation system from the graphical system, allowing simulations to run without the overhead of drawing the visual display (note that while running without the graphical system, it is still possible to use the networking support to connect to the simulation system from a separate computer and view the simulation from there)
- shares computer resources nicely when running in the background, permitting multitasking
- compatibility with Windows, Linux, or Solaris operating systems

One point that should be made at this stage is to discuss the default physics within the Quake 2 engine. Quake 2 does not support complex physical interactions, such as modelling a robotic arm picking up an object. Indeed, the built-in collision system treats all of the moveable objects in the world as rectangular boxes, although it is relatively easy (albeit slower) to modify the system to treat them as cylinders. The world geometry (the environment that the objects move around in), however, is modelled correctly. This limits the sort of simulation that can be easily done to things like basic solid-body collisions between a robot and any objects or walls it may encounter.

As far as sensory systems are concerned, Quake 2 supports a standard range-finder system. This consists of specifying a point and a direction (or two end-points), and the system will detect any solid object between these points. It is also possible to trace a solid volume along the line, rather than a single point. This is sufficient for implementing simulated sonar systems and bump (collision) sensors. It does not support vision sensors, although some similar effects can be done using the supported line-of-sight capabilities.

5.1.2 Software Design

The software architecture for the simulation is mainly constrained by the structure imposed by the Q2Java interface to Quake 2. In order to understand how the simulation works, a bit of detail on this system is required.

Quake 2 runs on a fixed synchronous time-step. That is, it updates the state of the world (and all of the objects within it) every tenth of a second (referred to as one “frame” – or, more accurately, as one “server frame”). Thus, in order to have code executed by the system, it is merely necessary to add the code to the list of functions to be called each frame (through a Java-style “listener” interface). For example, the controller for the robot (which sets its current velocity and heading) is registered with the engine to be called each frame, and so every tenth of a second a function in the controller will be run.

It is important to note that this does not actually impose a strict limitation on the programmer. At first glance, it seems that it would be impossible to write an asynchronous control system (one where things do not happen at a fixed time-step). In fact, it is possible to do this via Java's multithreading abilities. The asynchronous aspect of the system can be done in a separate thread (i.e. it would run independently of the Quake 2 system), and then every tenth of a second Quake 2 could update itself with the current values of that other thread. However, this sort of thing is not needed for the purposes of this paper, since the DAC algorithm (and, indeed, most control algorithms used in simulation) is synchronous (i.e. the entire system has a state at time t and a state at time $t+1$, but not at time $t+0.5$).

The modules of code (Java objects) which are run each frame can be any arbitrary Java code. However, some of them can be directly tied to objects in the simulation itself. For example, the code that controls the robot would be directly tied to the robot itself. This allows the code to manipulate the robot in various way. For example, it can change the robot's location within the world, or detect collisions between the robot and other objects (such as the targets in the DAC experiments). Furthermore, a special function is available to move the object within the world subject to physical constraints (i.e. collision detection). To do this, it is only necessary to specify the current velocity of the object, and the object will move accordingly. It is also straightforward to use this to deal with elastic collisions and gravity.

Q2Java provides two basic ways of introducing objects into the world. The first is through the map file, and the second is through the code itself. The map file specifies all of the environmental geometry of the world. That is, it specifies the locations of walls and floors (or highly complex sculptures). Generally, this geometry is non-moveable, although it is also possible to make simple moveable geometry in this manner, such as doors or moving platforms (neither of which are used for this thesis). The map file is also used to specify objects that should appear in the world. For example, in order to cause the four targets in the DAC experiment to appear in the world, the map file would contain four pieces of data that say "put an object controlled by the code in the target class here".

In order to facilitate the design of these map files, a number of different free software packages are available. They provide a graphical representation of the map file, and provide various high-level tools for producing complex environments (as well as dealing with lighting and the various textures one can apply to the walls – neither of which is important for simulation, but can be important for aesthetic purposes). The map files for my simulations were all made using QuArK (Quake Army Knife) [33]. The following image shows a top-down view of the DAC environment, modelled after the work done at the University of Zurich. Note the four small crosses which represent the locations where the targets will be created.

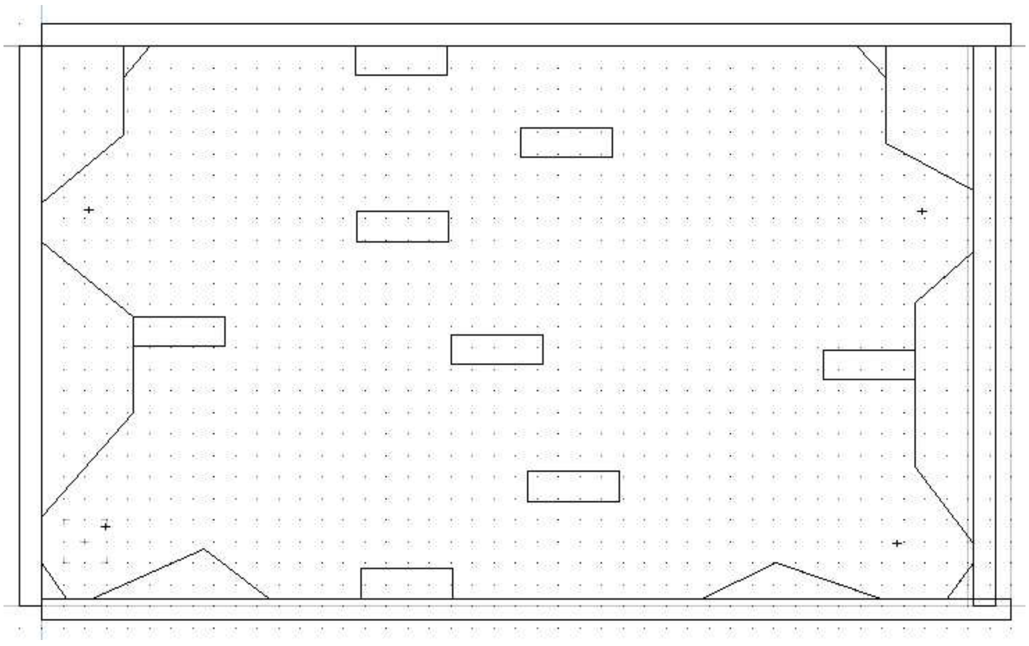

Figure 5.1: A top-down view of the DAC map file, showing the walls and the locations that the target objects will be created. Generated using QuArK [33]

We have not yet seen how the robot is actually inserted into this environment. This is done through the code itself. Q2Java allows for the creation of new "console commands" within Quake 2. Console commands are text commands which can be entered by any user connected to Quake 2. When the command is entered (for example, if a user types "start"), the specified code is run. This code can create new objects in the world simply by creating instances of the appropriate Java classes. These objects can be told their initial starting location and orientation (or any number of other arbitrary configuration commands, depending on the object itself). Once these objects are created, they can add themselves to the list of objects to be updated each frame. The visual representation of the robot is specified by referring to an external model file. Literally thousands of user-created models are available online: for this particular thesis, I chose a Dalek model available at [45]. Note that the visual representation of the robot does not in any way affect the simulation itself.

The final major component of the simulation software is the aspect which actually controls the experiment itself and records the results. This is another Java object which is called each server frame. In order to make it as easy as possible to run various different types of experiments, this experiment controller reads the experiment parameters from a text file. This file is simply a list of parameter names and their values. For example, it can specify the number of time steps that the experiment should run for, the speed of the robot, the number of collision sensors, the size of the robot, and even which Java class should be used to control the robot's movements. The experiment class uses this data to create the robot (with the specified controller). At each frame that the experiment is running, the class records various aspects of the robot's behaviour. This can include the robot's location, the total number of targets it has found, and even internal aspects of the control system, such as the state of individual neurons in the DAC system. All of this data is written to a file for later analysis.

Some readers may have noticed a significant problem with the simulation system as presented here. In particular, one of the advantages for simulation listed in the section on Simulation and Reality (see section 4.2) was that we can accelerate time in simulation to take advantage of faster processors. Since the objects in the simulation engine are updated once every tenth of a second, it seems that Quake 2 cannot do this acceleration. Fortunately, this problem is readily fixed. Since all of the source code for the Q2Java system is available, it was easy to find the part of the code which calls all of the objects each tenth of a second. It was then a simple matter of adding a new variable to the system which specified the number of times each object would be updated each tenth of a second. This then acts as a time multiplier (setting the value to 10 would cause each object to be updated 10 times every tenth of a second). Of course, if this value is increased past the capabilities of the system, then Quake 2 is no longer able to keep to its tenth-of-a-second schedule. Quake 2 handles this gracefully and simply updates as fast as it possibly can. For example, when doing the DAC experiments, setting the multiplier to 20 caused the system to update approximately 20 times every fifth of a second.

5.2 Duplication Experiments

In order to do any detailed experimentation with DAC, it is necessary to implement it and then to test that implementation. This testing takes the form of running the same experiments described in the Zurich papers. If the results are similar, then I will have both validated the software itself, and I will have provided evidence that DAC itself is reasonably robust. After all, it must be kept in mind that the physics of the simulation environment being used are not the same as those used in the original DAC experiments. If DAC is not robust, then the results achieved in the Zurich simulations may be strongly dependent on features in their particular simulation system, and so my implementation would not perform similarly.

In order to simplify the implementation, it is possible to split DAC into two stages. DAC0 is DAC without the learning system. That is, it only has the hard-wired connections from the collision and target sensors to the motors. It has no range-sensing ability, and its only behaviour consists of simple stimulus-response patterns (for example, if it hits a wall, it backs away and turns). DAC2 (which has the learning capabilities) can be built onto this in such a way as to not disturb the original DAC0 aspects. The first step, then, is to get a working version of DAC0.

5.2.1 DAC0

The implementation of the core DAC0 control system itself is not particularly difficult, since it simply consists of 37 collision sensors, each connected to one of two “back up and turn” motor outputs, and two target sensors connected to one of two “turn this way” motor nodes. A simple decaying inhibitory connection between the collision sensors and the target sensors is also needed. This completes the core wiring diagram for DAC0, as shown in the following image.

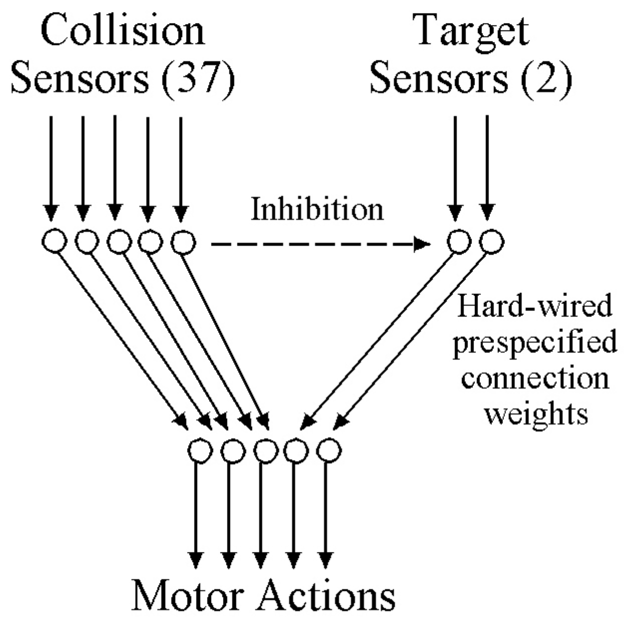

Figure 5.2: The basic DAC0 architecture.

There are, however, a number of other issues involved in fully implementing DAC0 into a simulation. We need to determine the robot’s movement velocity (how far it moves between time steps). We also need to determine its size relative to the environment. Fortunately, both of these values can be readily estimated from diagrams such as the following one.

Once all of these values are defined correctly, we can run the simulation. The robot is placed randomly into the environment and allowed to run for 5000 time steps (as in the original Zurich work). When this is done, we see that the robot moves around the environment and successfully finds the targets in a manner very similar to the paths shown in the Zurich papers. The results give the mean number of targets found as 30.225, with a sample deviation of 3.246. Unfortunately, there is no way of directly comparing this result to the Zurich experiments, since no quantitative data is available for runs of merely DAC0.

However, one worrying aspect is that occasionally the robot will get stuck by becoming backed

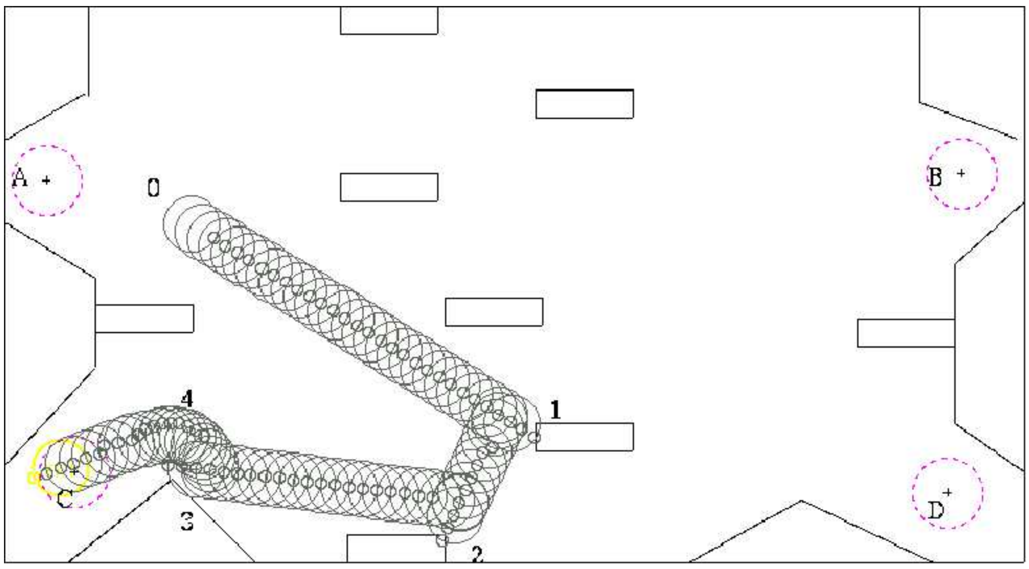

Figure 5.3: Figure 2C from [43], used to estimate the robot's size and speed.

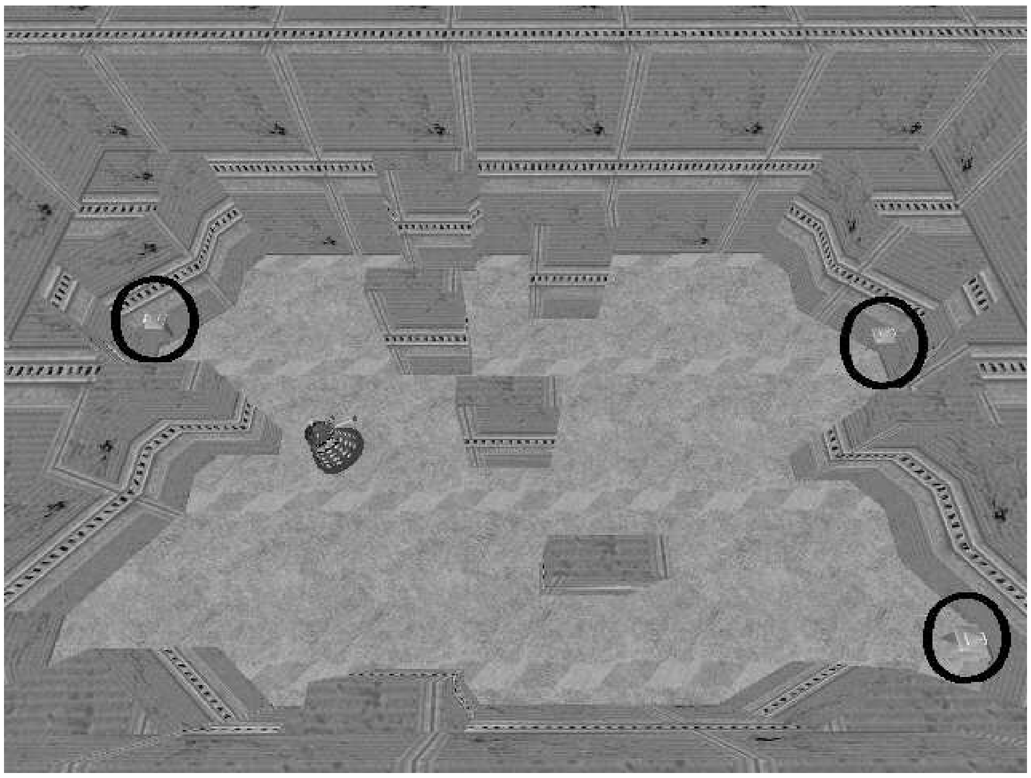

Figure 5.4: DAC0 running in the Quake 2 simulation environment. Note the three visible targets circled in black: the fourth target (normally in the bottom left) was previously picked up by the robot.

into a sharp corner. This happens because one collision sensor is triggered on one side of the robot, causing it to back up and turn. Since it is in a corner, physics does not allow the robot to back up

Figure 5.5: Another view of the simulation

through the walls, and so it merely turns in place. Now that it has turned, the collision sensor on the other side of the robot is activated and the cycle continues. This effect is not mentioned in the Zurich papers. It is possible that it is due to a difference between the two simulation environments used. However, it could also be that it is not mentioned because it only happens with DAC0. A robot running DAC2 may never (or seldom) get into such a situation, and so this behaviour would not be noted. Also, it was found that it is possible to eliminate this behaviour by rounding out the sharp corners of the environment.

Summary of Experiment:

- Time Steps: 5000
- Number of Collision Sensors: 37
- Number of Range Sensors: 0

Summary of Results:

- Targets Found: 31.7 (s.d. 1.78)
- Time Spent in Collisions: 1741.6 (s.d. 60.017)
- Number of Collisions: 225.25 (s.d. 9.176)

5.2.2 DAC2

Now that the very basics of the experimental system have been implemented, it is time to add the learning algorithm itself. This involves adding range sensors to the robot, a connection between the range sensors and the collision/target sensors, and implementing the learning method to adjust this connection. We can then run the system, count the number of targets found and collisions over time, and compare that to the results in [40].

Many questions arise when trying to add range sensors. For example, what should their maximum range be? Should they be linear? Should they only detect in a straight line, or should each range sensor handle a small angle? How much variation (noise) should there be in the response from the range sensor?

Fortunately, the Zurich papers do give the exact non-linear transfer function of the range sensors. This function takes the distance from the edge of the robot's body and converts it into a number between 0 and 1. Since the function drops off to zero as the distance gets larger and larger, the "maximum range" of the sensor is simply the distance at which the transfer function returns a value sufficiently close to zero.

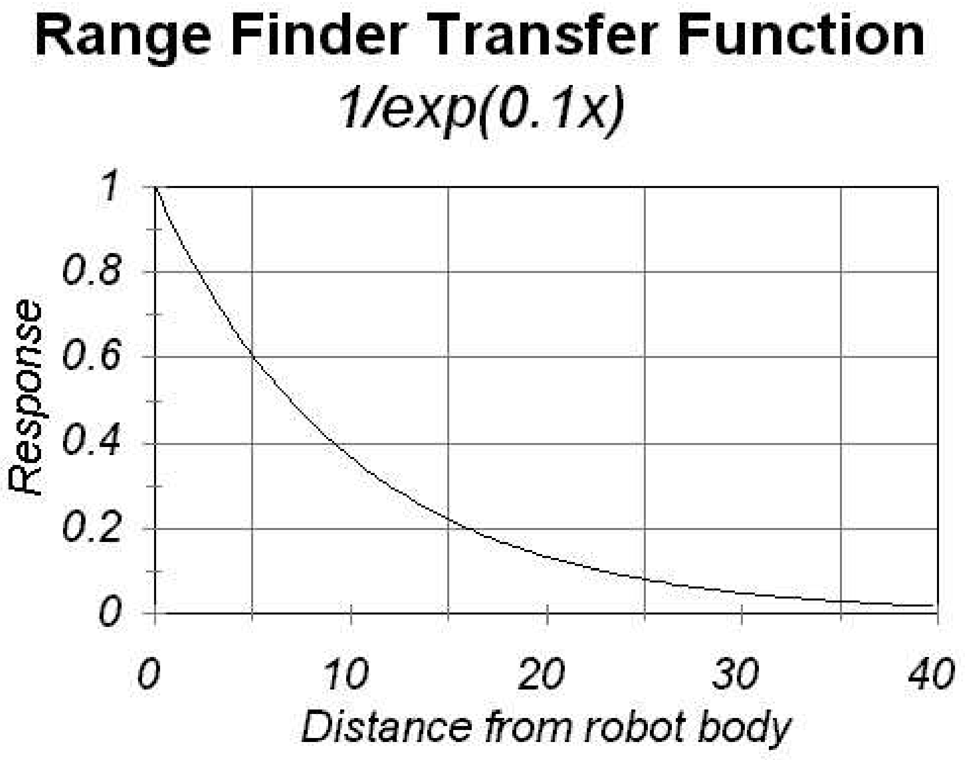

Figure 5.6: The response pattern for the range finders. A distance of 0 would be a solid obstacle touching the robot's outer body.

The diagrams in the Zurich work imply that each range sensor handles a small arc (approximately 5 degrees). However, there is no information on how much variation there is in the sensors, nor is there information on exactly how this region is handled (i.e. does the sensor respond to the average distance to an obstacle in the region, or the minimum distance?). For the purposes of this simulation, it was decided to answer both of these questions by randomly varying the exact orientation of the range sensor. For example, the range sensor at the very front of the robot would randomly choose an exact orientation (between -2.5 degrees and +2.5 degrees) at each time step. This adds both variability and a reasonable arc of sensitivity at a minimal computational cost.

The connections between the range sensors and the basic sensory system is simply an array of weights from each range sensor to each basic sensor. These weights are all initialized to zero. As in the Zurich work, the results of these weights are linearly added onto the output from the basic sensor itself. That is, even if a particular collision sensor is not active, its output is determined by the weighted sum of the inputs from the range sensors.

$$v_i^k = \sum_{j=1}^{N} w_{ij}^k u_j + c_i^k$$

Figure 5.7: Formula determining the output of a collision or target sensor. The output (*v*) is equal to the sum of the weights (*w*) from each range sensor to this sensor times the output of the range sensor (*u*), plus the actual activation value of this sensor (*c*).

In order to update the weights, we use the basic DAC2 learning algorithm. This updates the weights based on the difference between the current range sensor values (*u*) and the predicted value of the range sensors (*e*). The predicted value is based on the current output of the various basic sensors (*v*) and the same weights (*w*) as are used to allow the range sensors to affect the basic sensors.

$$e_j = \sum_{k=1}^{K} \sum_{i=1}^{M^k} w_{ij}^k v_i^k$$

$$\Delta w_{ij}^k = \eta^k v_i^k (u_j - e_j)$$

Figure 5.8: The DAC learning algorithm.

The predicted value of a range sensor (*e*) is the sum of all the outputs from the other sensors (*v*) weighted by the connection from the range sensor to that other sensor (*w*). The weight update is the difference between this prediction (*e*) and the actual value (*u*), times the activation of the sensor the weight connects to (v), times the learning rate (n). Also, the final value for each weight is constrained to be non-negative. Note that in the above formulas, k denotes the various different types of sensors (collision and target).

This leaves us with one further variable to specify. The "learning rate" is determined by how much the weights should change based on the current prediction difference. This is generally a variable in all weight-based learning systems, and is typically something around 0.01. In order to choose an exact value, we can run experiments with the learning rate set to various values. The best result from these experiments can be taken as the learning rate to generally use, as it is likely that the original DAC work was done using the optimal value.

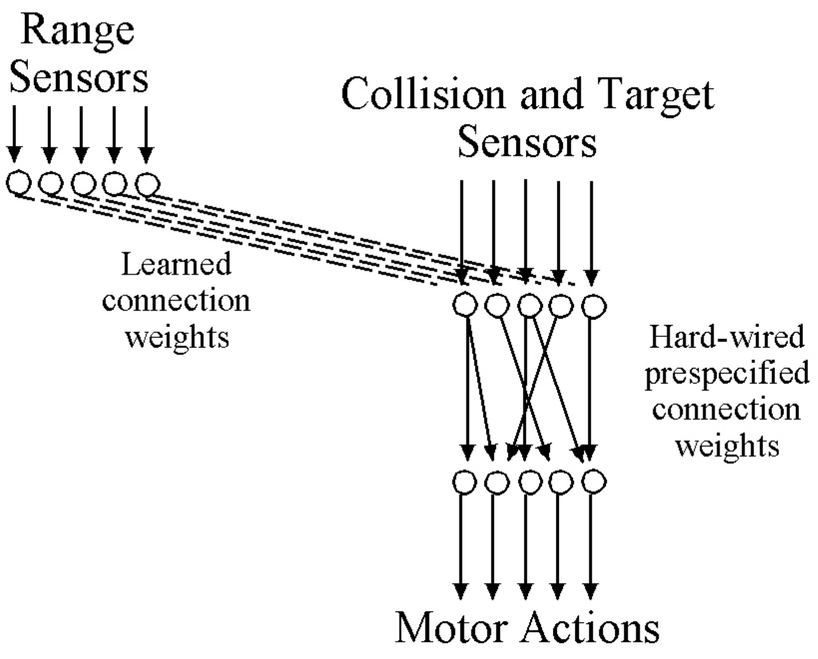
Range Sensors
Collision and Target Sensors
Learned connection weights
Hard-wired prespecified connection weights
Motor Actions

Figure 5.9: The DAC2 architecture.

Running these experiments requires us to store the number of collisions and the number of targets found. The number of targets found is easy to do, since the target is found at one particular time step and then disappears. This means that the robot does not encounter the same target again in the immediately following time step. However, this is not true for collisions. It is possible for the robot to have collided with an obstacle at one time step and then still be in contact with that obstacle at the next time step. Should this be counted as two collisions or one collision?

This is actually an interesting point which brings up the difference between real-world experiments (where there is no such thing as a "time-step" – or if there is, it's very very small) and simulation experiments. In the real world, with an experimenter observing a creature running around in an environment and colliding with walls, the experimenter is most likely to count each collision with a wall once, regardless of the amount of time spent in the collision itself. However, in the simulation world, it is easier (and perhaps more intuitive to a programmer) to count the number of time steps during which a collision is taking place. For example, if the creature comes into contact with a wall at a certain time, and then 5 time-steps later it manages to back away from the wall so that it is no longer in contact with it, then we would intuitively call this one collision. However, keeping track of collisions in this manner is more complicated from a programming perspective than simply adding one to variable every time-step that the creature is in contact with a wall. This second measure would lead to a much higher value for the "number of collisions". There is no clear evidence as to which definition of collision was used in the original Zurich papers. When using the first "collision count" definition, the results from my simulations give values at or below the Zurich group's few sample runs. When using the "collision time" definition, the results are generally above the Zurich results. In both cases, the discrepancy is approximately 1-2 standard deviations. However, the pattern of collisions over time (as depicted in the figure in the section on implementing DAC (see section 4.3.1)) matches better with the results using the "collision time" definition. The "collision count" approach does not grow as quickly in the initial time steps of the simlation. For clarity, I will be providing both measurements in this thesis.

The following graphs show the effect of varying the learning rate between 0 and 0.1. The three lines on each graph denote the mean value and plus-and-minus one standard deviation over 100 trials at each learning rate. It can be seen that DAC works reasonably well over this range. A value of 0.05 seems to be reasonably optimal for this particular task, and so will be used for the rest of this thesis.

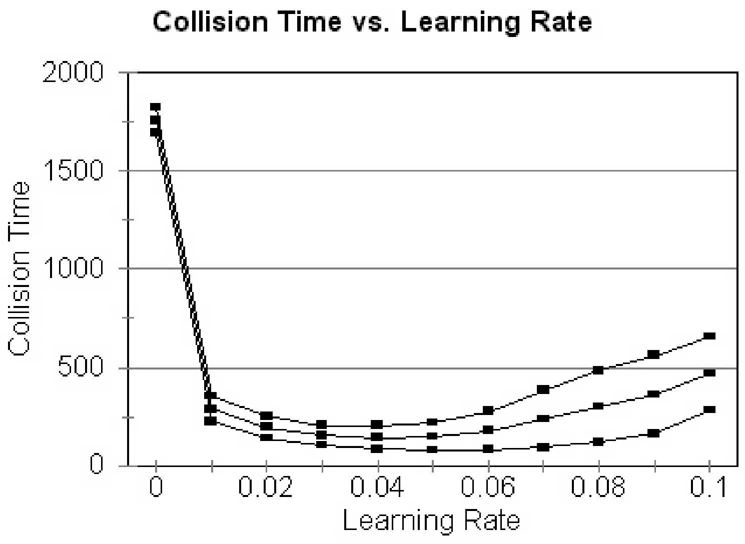

Figure 5.10: Graph of amount of time spent in collisions for various different learning rates. Mean and plus/minus one standard deviation are shown.

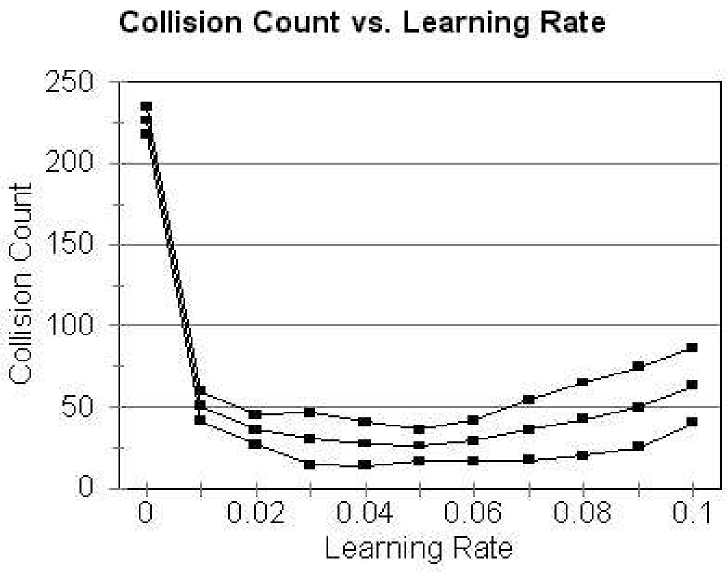

Figure 5.11: Graph of number of collisions for various different learning rates. Mean and plus/minus one standard deviation are shown.

With a learning rate of 0.05, the mean number of collisions by time is 150.11 (sample deviation 71.982) and the mean number of targets found is 33.99 (sample deviation 3.335). This is not the same as the example run given in [40], where the number of collisions is 45 and the number of targets found is also 45. However, this result is within the bounds of variability. In fact, one particular run gave a collision time of 23 and a target count of 45. Thus, this simulation can fairly

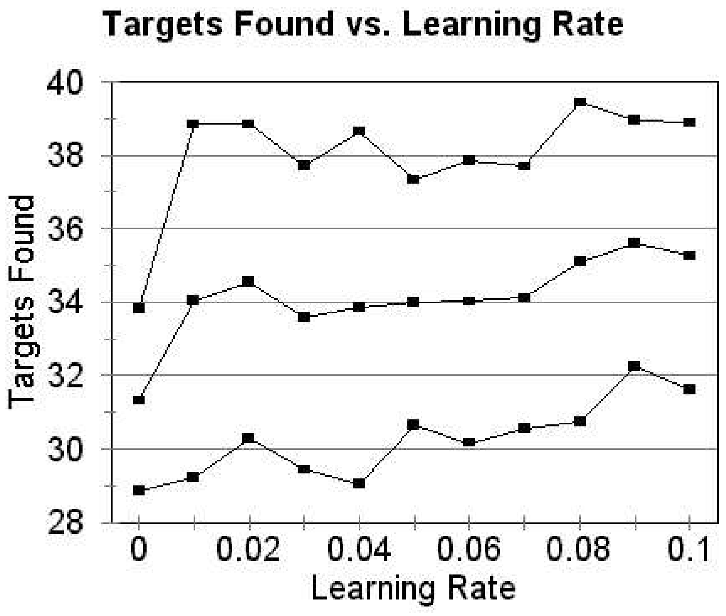

Figure 5.12: Graph of number of targets found for various different learning rates. Mean and plus/minus one standard deviation are shown.

easily give results similar to the one sample run given in the initial paper. It should be remembered, however, that the sample run is one fairly good run, not an average run. It should also be noted that the robot no longer gets stuck in corners as it did with DAC0.

With these experiments done, it seems that the implementation of DAC is faithful to the original algorithm. The differences between these results and those published by the Zurich group could be reasonably explained by the differences in the exact physics of the two simulation systems. Also, the data available on DAC tends to discuss example runs of the system, rather than averages over many runs, making it difficult to perform any sort of precise numerical comparison. However, the system is clearly qualitatively performing the kind of learning that DAC is supposed to do. We can thus go on and perform the other experiments we are interested in, reasonably confident that this implementation of DAC works.

Summary of Experiment:

- Time Steps: 5000
- Number of Collision Sensors: 37
- Number of Range Sensors: 37
- Best Learning Rate: 0.05

Summary of Results:

- Targets Found: 33.99 (s.d. 3.335)

- Time Spent in Collisions: 150.11 (s.d. 71.982)
- Number of Collisions: 26.1 (s.d. 9.73)

5.3 Ancillary Experiments

Before going on to the classical conditioning experiments, however, there are two mini-experiments which were performed on the DAC2 system. These were done to study various aspects of DAC learning. The first looks at what happens when the robot loses its ability to sense targets, and the second looks at the effects of varying the number of sensors the agent has.

5.3.1 Target Sensing Experiment

Most of the analysis of the DAC system focuses on its ability to form associations between the range sensors and the collision sensors. However, very little is said about the associations between the range sensors and the target sensors. To determine if there is any useful learning occurring in this area, an experiment was performed which compares the robot's behaviour with and without learning when it suddenly loses its ability to sense targets.

This experiment is inspired by experiments performed more recently by the Zurich group while developing DAC3. They used the loss of target sensing ability as a forum for testing their addition of the learning of temporal chains to DAC. In [43], they perform tests by having the system run normally for 2000 time steps, then removing the target sensor ability for 5000 time steps. They then counted the number of targets and the number of collisions for DAC0, DAC2, and their DAC3. Oddly, the results for DAC2 were that it found significantly fewer targets than DAC0 or DAC3 (34 for DAC2 versus 53 for both DAC0 and DAC3), indicating that its ability to learn hampered it in the task of finding targets (of course, DAC2 collided with significantly fewer walls than DAC0). This indicates that not only does learning not help it in the task of finding targets, but that it in fact significantly hinders it. This could, perhaps, be due to the fact that the targets in the environment are somewhat near to the walls, so after learning to avoid walls it spends less time near targets as well.

The results from performing the same experiment with my simulation are that there is no significant change in number of targets found between DAC0 and DAC2. DAC2 found insignificantly fewer targets (29 versus DAC0's 31), and collided significantly less often. From this, we can make two tentative conclusions. Firstly, there is no evidence that learning between the range sensors and the target sensors is affecting the robot's behaviour in any way. Secondly, there seems to be some significant variation between my simulation and the Zurich simulation. This could take many forms, but it is perhaps relevant to point out that new experiment was described in a 1999 publication, while the rest of the work that this thesis is based on was published in the 1992-1994 range. In the intervening time, it is possible that various features of the Zurich simulation have changed.

Summary of Experiment:

- Time Steps: 7000
- Stop Sensing Targets After: 2000 time steps
- Number of Collision Sensors: 37
- Number of Range Sensors: 37

Summary of Results:

- DAC0:
  - Targets Found: 31.15 (s.d. 3.167)
  - Time Spent in Collisions: 2497.9 (s.d. 72.935)
  - Number of Collisions: 318.1 (s.d. 9.878)
- DAC2:
  - Targets Found: 28.9 (s.d. 3.582)
  - Time Spent in Collisions: 184.9 (s.d. 102.678)
  - Number of Collisions: 33.45 (s.d. 14.816)

5.3.2 Morphology Experiment

The second experiment starts to question the pattern-finding abilities of DAC. In particular, it focuses on the simplistic mapping from range sensors to collision sensors in the standard DAC configuration. Because there are exactly the same number of collision sensors as target sensors, and since they are lined up in exactly the same way, the patterns that DAC needs to find are simple one-to-one mappings between each range sensor and its corresponding collision sensor. Since this is the only sort of configuration used in the DAC experiments, it may be that DAC is incapable of discovering more complex patterns. So, to test DAC's abilities, this experiment varies the number of target and range sensors (its sensor morphology) and examines the effects on the number of collisions and number of targets found.

The values for the number of sensors were chosen to have few common divisors, so as to ensure complex relationships between the sensor systems. The chosen values were 6, 13, 21, 29, 37, and 45. Each combination of settings was run through the standard test of running for 5000 time steps and counting the targets found and the number of collisions. The mean results for 30 runs are shown in the following charts.

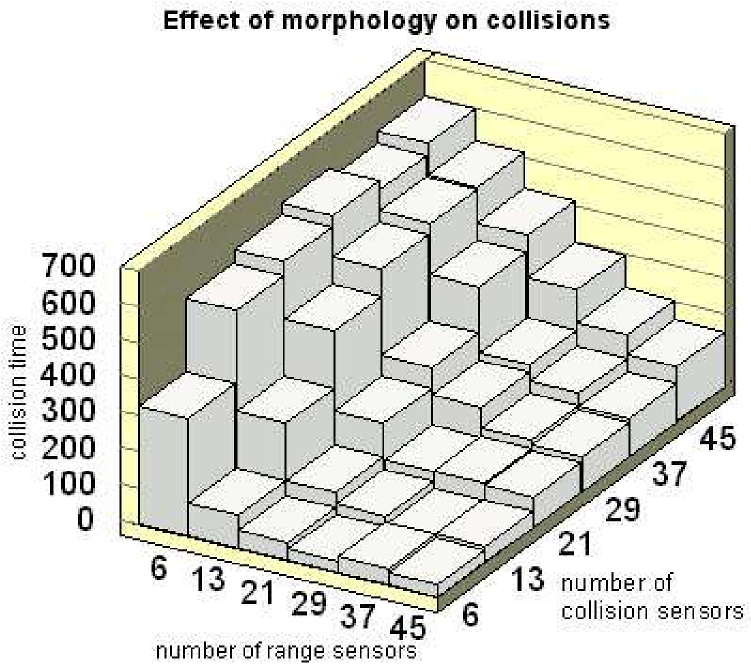

Figure 5.13: Amount of time spent in collisions for different numbers of range and collision sensors.

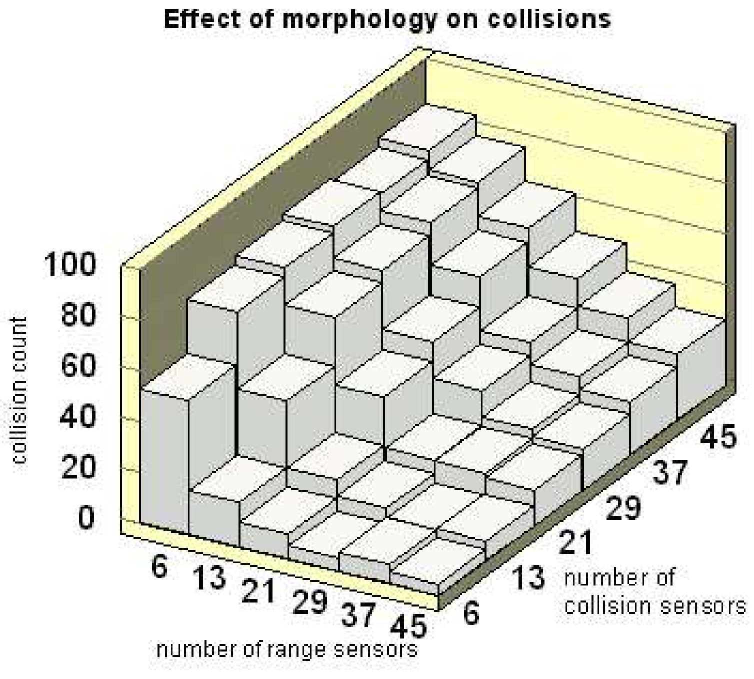

Figure 5.14: Number of collisions for different numbers of range and collision sensors.

The first observation from these results is that DAC does not require the same number of range sensors as collision sensors. For example, in the situation with 6 collision sensors and 37 range sensors, the system does successfully learn to avoid obstacles while still finding the same number

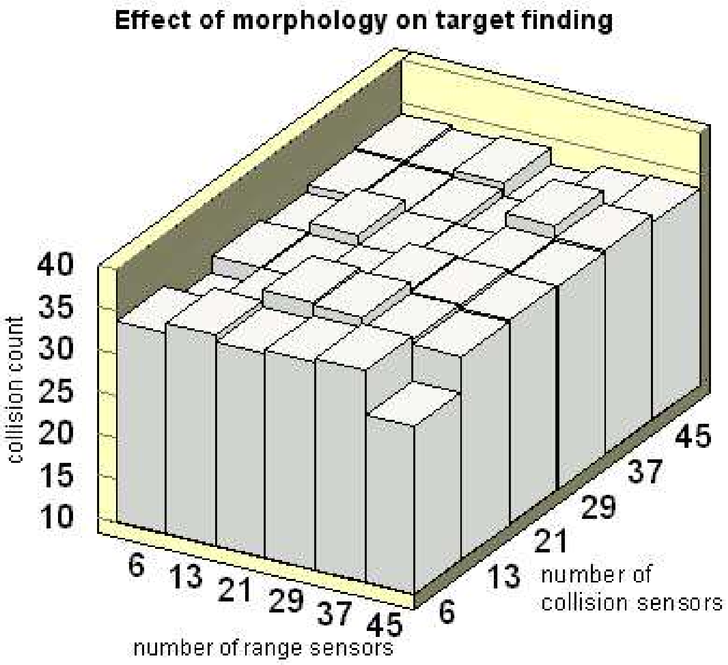

Figure 5.15: Number of targets found for different numbers of range and collision sensors.

of targets as the standard system with 37 collision sensors. This means that there is no one-to-one mapping limitation, which is a great relief, as one of the major design principles behind DAC is to form associations between patterns, not individual sensors.

The second observation is a more interesting one. First of all, we can see that there is no significant variation in the number of targets found for all of these configurations. However, there is a very striking pattern in the number of collisions. In particular, the system performs significantly better when there are more range sensors than collision sensors. Conversely, having fewer range sensors, performance is impaired. This means that reducing the number of collision sensors improves the agent's performance.

This is a somewhat non-intuitive result, but it is, in hindsight, explainable. This can be done by observing that the task that the robot is performing does not actually require more than two collision sensors. If it had just one large-angle collision sensor on the left and one large-angle collision sensor on the right, then the system would still work perfectly well. Thus, reducing the number of collision sensors does not reduce the robot's theoretical ability to perform in any way. However, reducing the number of collision sensors does reduce the number of weights in the system that need to be learned. Fewer collision sensors leads to a less complex system, which should thus be easier to learn.

Another (admittedly more speculative) way of viewing this phenomenon would be to appeal to Ashby's Law of Requisite Variety [1]. What is actually happening here is that the system is using the range sensors to predict the values of the collision sensors. If there are more range sensors than collision sensors, then there are more degrees of freedom available. Ashby's Law implies that a system with a larger number of degrees of freedom has a much easier time controlling or predicting a system with fewer degrees of freedom. Conversely, it is very difficult to use six range sensor values to predict 37 collision sensor values.

Chapter 6

Conditioning Experiments

Now that we have gained some experience with the DAC simulation developed for this thesis, it is time to turn to the important experiments which can determine the characteristics of the DAC learning algorithm. The hypothesis expressed so far is that we do have, in DAC, an effective model which explains associative learning in a way consistent with classical conditioning. However, the original Zurich experiments have only shown that DAC is capable of a particular aspect of classical conditioning. This aspect, the ability to form an association between a new stimulus pattern and an old stimulus pattern, is at the core of classical conditioning, but it is not the full story. A hundred years of psychology experiments have uncovered many other quirks and characteristics of classical conditioning, each of which may well be vital to explaining the effectiveness of this style of learning. For example, the ability to acquire associations may not be very useful without the ability to have these associations go away (extinguish) over time if the learned pattern is no longer applicable. Does DAC have this characteristic? It is never tested in the Zurich experiments.

Answering these, and other, questions about DAC's real abilities is the goal of this section. I will first describe the particular aspects of classical conditioning which need to be evaluated, and then detail the experiments performed.

6.1 Overview of Conditioning Test Suite

The tests for determining how well DAC matches with classical conditioning can be divided into two parts. The first part explores DAC's concept formation abilities, and the second part examines the more traditional aspects of classical conditioning: acquisition, extinction, generalization, and specialization. For the purposes of this thesis, blocking and second order conditioning were not examined. This is because these two features were explicitly part of the design of the DAC learning model, as described earlier (see section 4.3). The intent here is to examine the other aspects of the DAC model, in order to evaluate the match between it and natural classical conditioning.

The concept formation experiments are, in my view, the most interesting. Firstly, they address a part of learning which does not get much attention either in Artificial Life research or in Psychology research. In most models of classical conditioning, the concepts do not need to be extracted from the sensor data, since the sensor data directly presents those concepts that the agent is supposed to learn about. Secondly, the very process of determining if a concept is formed is problematic. How can we evaluate whether the agent has formed a particular concept? This is a question about mental states and the internal workings of the agent's "mind". Even with DAC's simple architecture, it is not easy to analyse the resulting neural network to determine the presence of various concepts. Indeed, according to the dynamical systems view, these concepts may not be found by looking solely at the organism's internals; instead, the outside environment must be examined as well. Thus, we can only answer this question by looking at the actions of the agent within its environment. The concept formation experiments revolve around a test which operationalizes the definition of a "concept" in a manner that is based on potential utility to the agent. In other words, it is based on the agent being able to make predictions about its environment. The difficulty of doing these experiments and in making strong conclusions based on the results points out many of the intrinsic issues with trying to do experimental science with something as nebulous as a "concept". However, the tests definitely do provide insight into DAC's effectiveness as a learning method for various tasks.

The further experiments which look at various standard features of classical conditioning are very typical experiments, based directly on those found in undergraduate psychology textbooks [2]. The only difference here is that they are being applied to a simulated organism, rather than an organic one. The results are thus directly comparable to the expected results in behavioural science. Deviations from the "natural" results indicate the areas in which DAC does not match well with real classical conditioning.

It is hoped that these tests are not solely applicable to DAC. Each experiment is described in terms of stimuli and responses, so that it would be in theory possible to run the same set of tests on a different learning algorithm. This could provide a solid basis of comparison for researchers trying to emulate classical conditioning. As far as I am aware, no other test suite of this sort exists.

6.2 Concept Formation Experiments

There is no (known) way of dissecting a human's brain and analysing it to determine if they have the concept of, say, "fire", for example. However, we are quite ready to believe that people do have internal representations of such concepts. This is because, when we observe individuals behaving in an environment, they act as if they have certain concepts. People act as if they know that a fire on one side of a room has similar properties to a fire on another side of the room. They can assume that a new fire that they see will also be "hot", given that fires they've met in the past have also been “hot”. Thus, the determination as to whether a certain concept exists is done by examining the behaviour of the organism. This side-steps the issue of how concepts are represented in the brain by providing a functional test for the concept.

The question is, however, how do we do this with DAC? What sort of test can we do to see if the agent behaves as if it has a certain concept? To start with a simple concept, how can we test to see if it can have the concept of “being near a wall”?

One way to do this would be to see if it could form an association between “being near a wall” and some unconditioned stimulus. We can change the question from “does it have this particular concept?” to “can it perform some consistent action whenever confronted with a situation that we would identify as a particular concept?” This is something which can be directly tested through an experiment.

We do this by introducing a new unconditioned stimulus-response pair. This is a very simplistic S-R configuration, with one neuron for the stimulus, which is directly wired to the one neuron response. This can be thought of in a similar manner to the presence of food causing drooling, or any other standard unconditioned situation. Now, we allow the robot to explore its environment, and whenever it encounters whatever concept we wish it to learn, we activate this new stimulus. For example, whenever the robot comes “near a wall”, we can trigger the unconditioned stimulus, which fires the unconditioned response, which we can record. We now run our experiment to see if the robot can learn associations based on this concept. In other words, can it use its other sensors (i.e. the range sensors) to predict this concept of being near a wall?

The advantage of this test is that it can be applied to any concept that an experimenter can dream up. This is also a disadvantage, since many of these concepts will be completely impossible for the robot to work with. For example, “being near something blue” would not be something that the basic DAC system could handle, since it has no colour sensing ability.

However, it is important to note that the system would not necessarily fail if tested on “being near something blue”. It is quite conceivable that it could seize upon some other regularity in the environment that the experimentor did not expect. For example, if blue objects tend, in the particular environment being tested, to be out in the open with few other objects around them, then DAC may well be able to form associations based on that regularity. DAC would thus appear to be learning the concept of “being near something blue”, while it was actually forming some other pattern.

This observation is not a problem for this set of experiments. Indeed, it merely points out a truth in all experimentation based on concepts: there is no guarantee that the concept the experimentor is using matches perfectly with the concept that the organism is using. This is simply something which should be kept in mind while performing and interpreting these experiments. For a more theoretical discussion of these issues, the reader is referred back to the section “On Concepts and Existence” (see section 2.5).

In any case, interpreting these tests is a complex task in and of itself. At any given moment, the new stimulus sensor could be active or not active, and at the same time the agent either predicts or does not predict that the sensor should be active. There are thus four distinct possibilities that may happen at each time step. We call these “True Negative”, “True Positive”, “False Negative”, and “False Positive”, where “True” and “False” denote whether the agent is correct in its predictions, and “Negative” and “Positive” denote the agent’s prediction. This terminology is taken from the standard experiments with supervised neural networks.

From any particular experimental run, we merely count the number of time steps the agent is in each of the four states. The desired result would be large values for True Negative and True Positive and small values for the two False states. The situation is complicated by the fact that we cannot control how often the agent should actually have its sensor activated. For example, if the concept under experimentation is “stuck in a corner” and the agent never gets close enough to a corner to be considered stuck, then there can be no True Positive states (or, indeed, False Negative states). The agent could simply never learn anything and would end up having a large True Negative count and nothing in the other states. In order to watch out for this sort of problem, it is necessary to make sure that the experimental configuration is such that the robot does spend a significant amount of time both with its “concept” sensor active and with it not active.

One possible way of combining these measurements to make a quantitative evaluation as to whether a concept has been learned would be simply to take the ratio of True states to False states. This would determine what percentage of the time the agent is correct about its prediction. This is, however, not a particularly useful measurement, in light of the above observation about how often the sensor itself is active. With a simple measurement of how often the agent is correct, then an experiment where the sensor should be active 25 percent of the time would be successfully completed by an agent which did not learn anything. By simply always predicting the sensor should be off, the agent would be correct 75 percent of the time.

We can get around this problem to some degree by looking at two separate measurements: the Positive Accuracy and the Negative Accuracy. The Positive Accuracy is defined to be the ratio of the True Positive to the total of True Positive and False Negative (mathematically TP/(TP+FN)). In other words, it is a measurement of how often the prediction is correct when the concept sensor should be active. The Negative Accuracy is the opposite, defined by TN/(TN+FP). If both of these ratios are above 0.5, then it can be clearly said that the agent has a representation of the concept which works more that fifty percent of the time. That is, if the sensor should be active, then it is predicted to be active more than half of the time, and if it should not be active than it is predicted to be not active more than half of the time. If the first ratio is below 0.5, then the agent is under-sensitive to the concept, and if the second ratio is below 0.5, then the agent is over-sensitive to the concept.

It is important to realise that being over-sensitive to a concept is not necessarily a bad thing, especially if the concept is rarely encountered in the environment. Indeed, it could be argued that the original DAC experiment's success is due to the fact that it is over-sensitive to the concept of "colliding with a wall". Once it has learned its obstacle avoidance behaviour, it predicts collisions even when the robot is a few steps away from the obstacle. This is then a False Positive prediction, but it is still useful to the agent.

Indeed, even if the True Positive count is less than the False Positive count, we should not say that the concept has not been learned. This is a non-intuitive statement, since if there are more False Positives than True Positives, then a positive prediction will, on average, be a false prediction. However, if the two afore-mentioned ratios are still above 0.5, then it can still be argued that the concept has been learned, since the agent has a better-than-chance ability to predict the actual value of the concept both when it is there and when it is not there.

The actual concept formation experiments were done using the standard DAC architecture, with the addition of the one new sensor which signals the concept to be learned. This one sensor was directly wired to an output node which can be experimentally recorded. The environment used was a simpler one than the DAC experiments: a single room with four enclosing walls, approximately the same size as the environment in the original experiments. The agent is allowed to explore the environment over 5000 time steps, and a count is made of the four possible states: True and False Positive and Negative. The agent's current prediction of the sensor value is made based on what the output of the concept response node would be based solely on the range sensor (conditioned) stimulus. This gets around a standard problem in classical conditioning experiments, where it is generally not possible to tell if the creature's response is due to the conditioned or unconditioned stimuli. As in the original DAC experiments, the threshold for the agent's response is 0.5.

Due to the simple nature of the sensors being used, very simple concepts are required. The first concept is simply being near a wall. In this experiment, the concept sensor is activated if the agent is within a certain distance of any wall. Setting this specific distance to various values was also examined. Because the standard morphology for the agent is such that it only has sensors which cover the front half of its body, this concept of being near a wall is not trivial for the agent to discover, and indeed it is impossible for it to learn this concept one hundred percent correctly. This is because the agent cannot distinguish on the basis of forward-pointing range sensors the case of having its back to a wall from being out in the open. Note that this is a situation where DAC's lack of temporal memory causes real problems. This experiment was also carried out after modifying the agent to have sensors which cover 270 degrees – giving it the ability to see behind itself.

The same style of experiment was also done for corners (where the sensor would be activated if the agent was sufficiently close to two walls). Experiments were also done where the sensor was activated whenever the agent was in a narrow corridor which divided two rooms. Further experiments were done with the opposite of these three concepts. In these cases, the sensor would be active at the times when it was non-active in the previous experiments, and vice versa.

It should be noted that the task the robot must solve is not as trivial as it may seem. Since the DAC model has no temporal memory, the robot must make a judgement between its being in a corner and being in a corridor based on a single moment's worth of data. Furthermore, its sensors are affected by random variation, and thus do not provide perfectly reliable data. For comparison, in [22], a hand-designed algorithm for distinguishing walls from cylinders achieved 35 percent accuracy and an evolutionary algorithm achieved 70 percent accuracy.

6.2.1 Results

The following diagrams show the results of running the described experiments. For each configuration, 10 experimental runs were made, and the mean values for each measurement were taken. The first graph within each set of three gives the amount of time (out of a total of 5000 time units) spent performing the different possible types of predictions: True Negative (TN), True Positive (TP), False Negative (FN), and False Positive (FP). The second graph combines TN and TP into a total count of the correct predictions (True) and incorrect predictions (False). The third graph gives the agent's chance of forming a correct prediction if the concept in question is present (Positive) or absent (Negative).

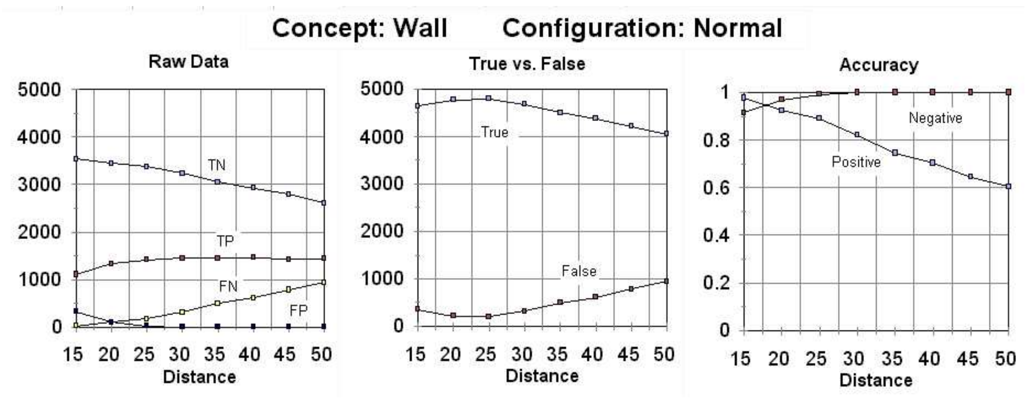

Figure 6.1: Results for learning the "wall" concept at various distances with 180 degree view.

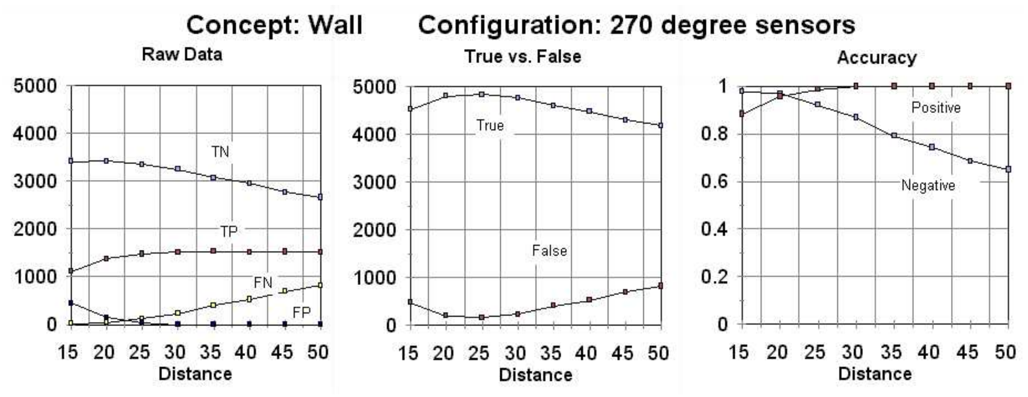

Figure 6.2: Results for learning the "wall" concept at various distances with 270 degree view.

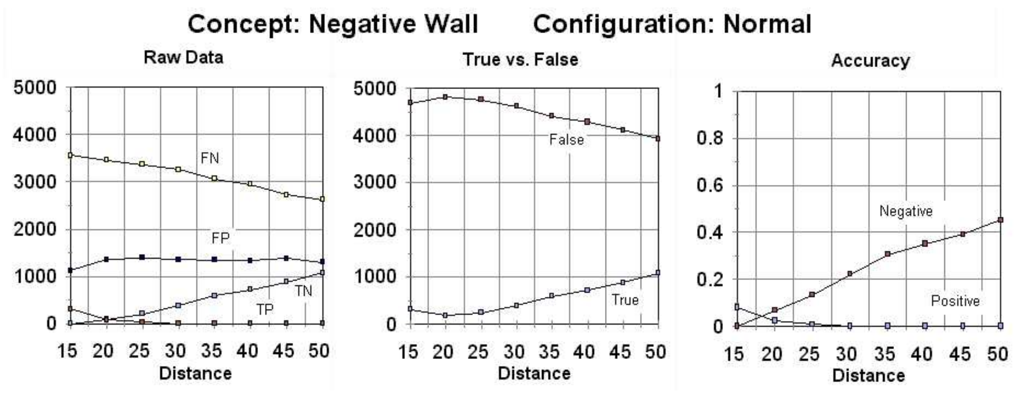

Figure 6.3: Results for learning the negative of the "wall" concept at various distances with 180 degree view.

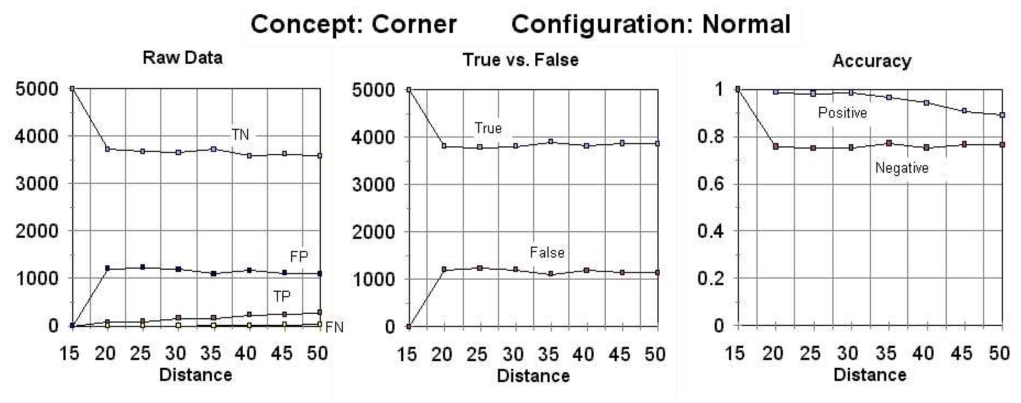

Figure 6.4: Results for learning the "corner" concept at various distances with 180 degree view.

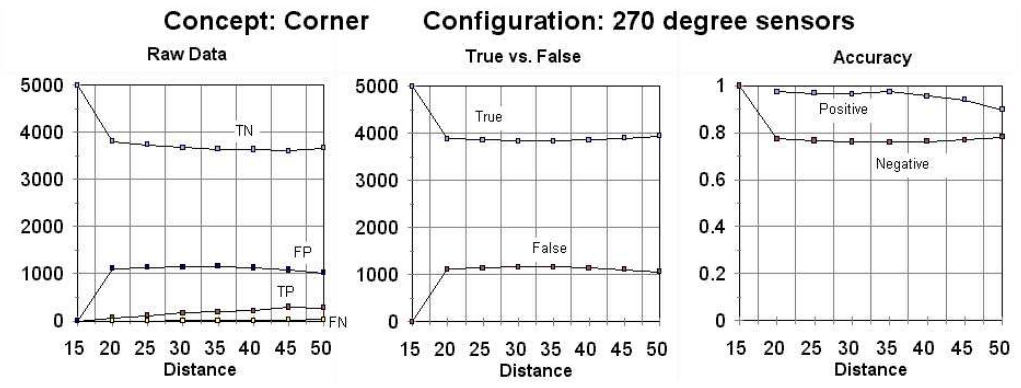

Figure 6.5: Results for learning the "corner" concept at various distances with 270 degree view.

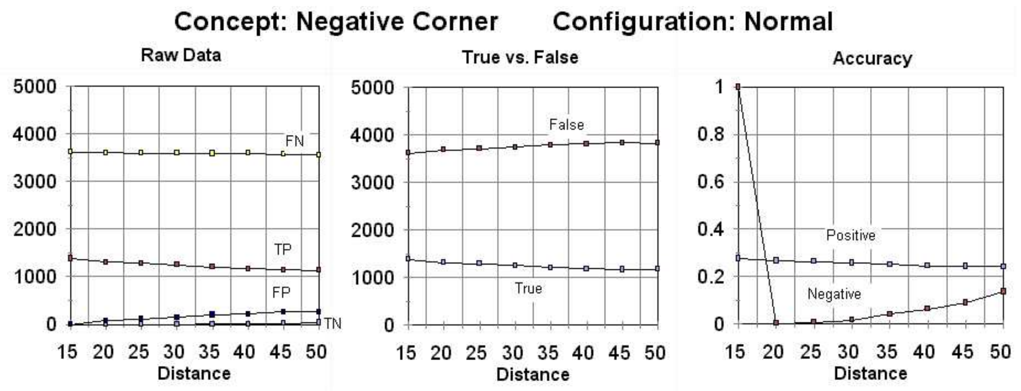

Figure 6.6: Results for learning the negative of the "corner" concept at various distances with 180 degree view.

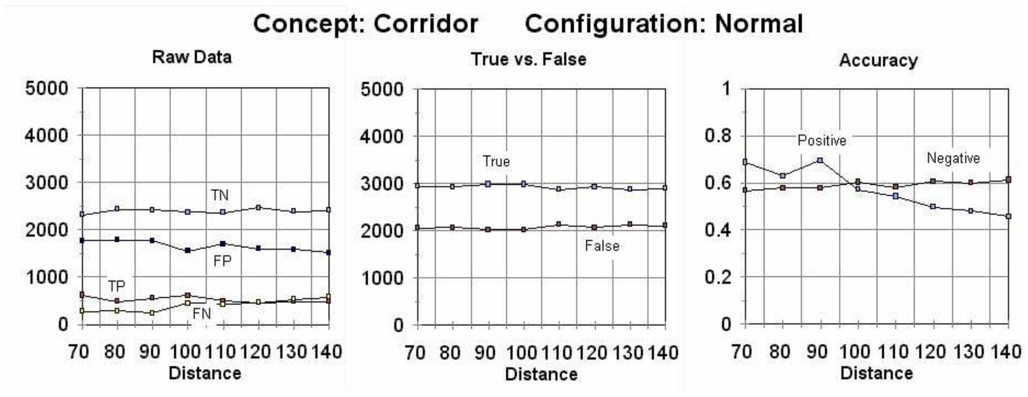

Figure 6.7: Results for learning the "corridor" concept at various distances with 180 degree view.

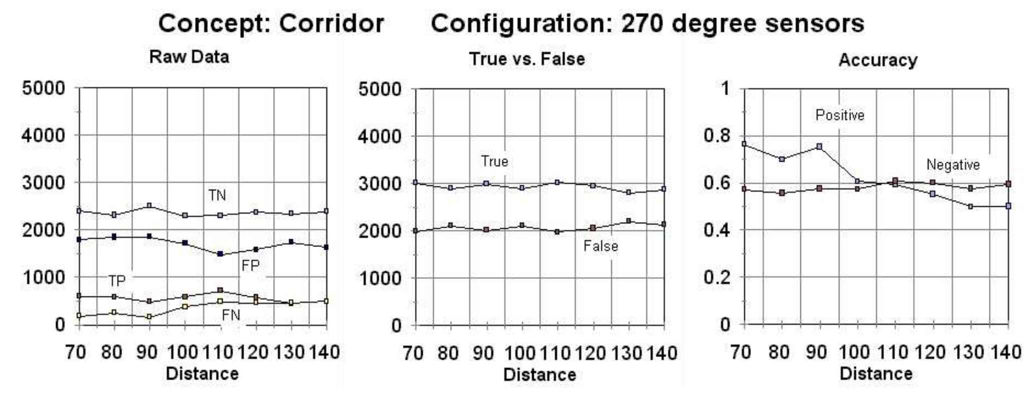

Figure 6.8: Results for learning the "corridor" concept at various distances with 270 degree view.

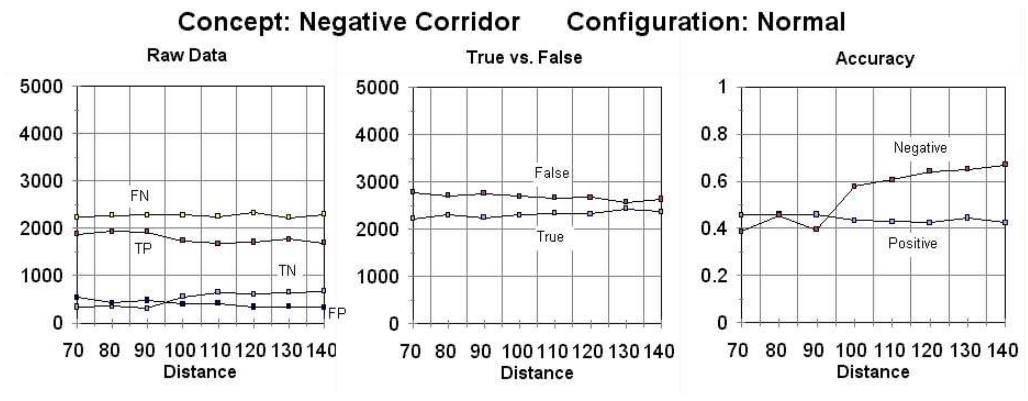

Figure 6.9: Results for learning the negative of the "corridor" concept at various distances with 180 degree view.

From these results, we can see that the system is quite capable of forming a better-than-chance ability to recognize the concepts of "walls", "corners", and "corridors". Other than a slight difficulty with detecting very wide corridors, where the accuracy drops to a minimum of 0.46, DAC's accuracy is comfortably above 0.5 for both the presence and absense of these basic concepts (see the final graph in figures 6.1, 6.4, and 6.7). We can also see that expanding the agent's field of view to 270 degrees slightly improves the performance of the system, as expected (see the final graph in figures 6.2, 6.5, and 6.8). However, a limitation of DAC is revealed in its inability to learn the reverse (negative) of these concepts. For these reversed situations, DAC performs substantially worse. In fact, its accuracy for these concepts (see figures 6.3, 6.6, and 6.9) is approximately 1 minus its accuracy for the basic concept.

This difficulty is not uncommon to learning systems of this type, however. DAC is closely related to Hebbian learning, and inherits its property of associating large values with other large values. In other words, it cannot form an association between one value being large and another being small. In statistical terms, this style of learning works solely on positive correlations, not negative ones. This is an important limitation on DAC's abilities, and one not pointed out by the initial research.

In an attempt to determine if this is the only problem with learning these negative concepts, an experiment was done exactly as above, but with the range sensors reversed. That is, their output was set to be 1 minus their normal output (bounded by 0 and 1). This resulted in an improved recognition ability, but even with this improvement, the system's performance was not as good as on the original concepts. The following three charts show DAC's learned response for these three configurations of the wall concept test.

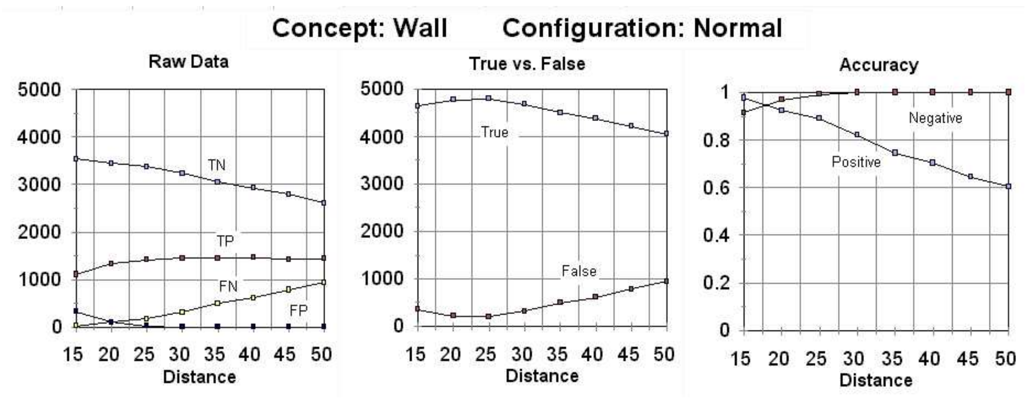

Figure 6.10: Results for learning the "wall" concept with 180 degree view. (repeat of figure 6.1)

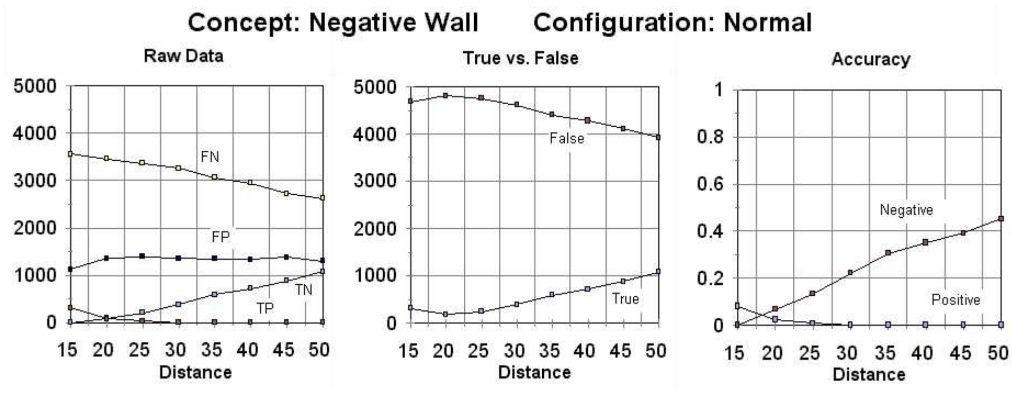

Figure 6.11: Results for learning the negative of the "wall" concept with 180 degree view. (repeat of figure 6.3)

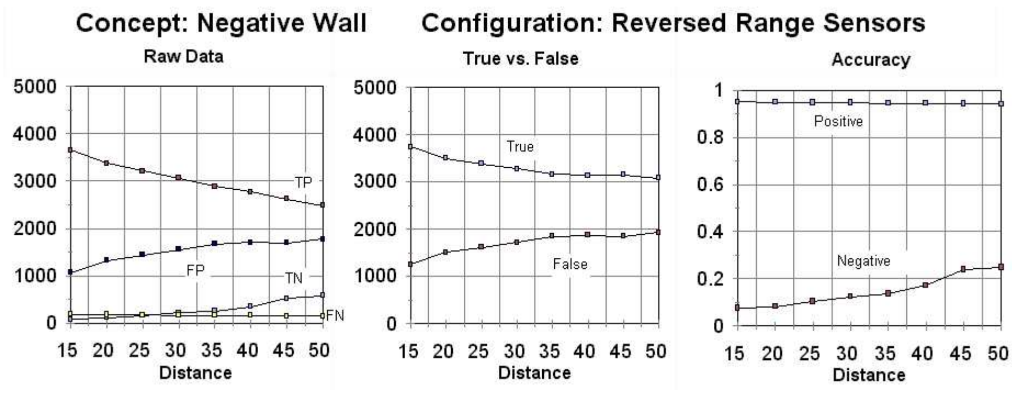

Figure 6.12: Results for learning the negative of the "wall" concept with 180 degree view and reversed range sensors.

It is also interesting to compare DAC's ability in these three configurations with that of a simple perceptron (two-layer neural network). By doing this, we see that there is definitely sufficient information in the range sensor data for the learning system to find. This means that there is definitely a statistical correlation between the two types of sensory data. This is important as it confirms that the task is, in fact, possible, and we are not expecting the system to learn something which cannot even in theory be extracted from the provided data. Of course, we should also expect that a perceptron would outperform DAC, since DAC does not have the benefit of multiple presentations of the data set. In fact, the perceptron performs identically on the three configurations mentioned above, and in all cases gives results slightly better than DAC does for the basic "wall" concept.

6.3 Features of Conditioning Experiments

The following experiments specifically examine particular features of classical conditioning. Since these are to be directly compared to results derived from experiments on living creatures, it is important to make the DAC situation as close as possible to the real-life situation. For this reason, these experiments do not use stimuli that are patterns to the agent, but rather use a local representation. That is, associations are formed between single sensor nodes. This is because the live experiments do not consider the concept formation requirements of classical conditioning in any way. They consider something like "hearing the sound of a bell ringing" to be a single sensory input. We have discussed previously (see section 4.3) how this is not a reasonable assumption, but for the purposes of this section we will make it.

6.3.1 Acquisition

There are three standard acquisition methods used in psychology: simultaneous, delayed, and trace conditioning. The difference between these three methods is in the timing of the presentation of the conditioned and unconditioned stimuli, as shown in the following figure.

Thus far in this paper, we have only considered simultaneous conditioning, where the stimuli that are being associated are always present at the same time. However, both trace and delayed conditioning are apparent in living creatures. In this experiment, we compare DAC's ability to form associations in these three manners.

This is done quite simply by directly controlling a sensor value for the unconditioned stimulus and the conditioned stimulus. Note that the actual behaviour of the agent does not affect this experiment in any way (other than the one output tied to the unconditioned stimulus). This is to make the situation as close as possible to a standard psychology experiment; the actions of Pavlov's dogs do not affect when the stimuli are presented. This is merely part of the experimental configuration.

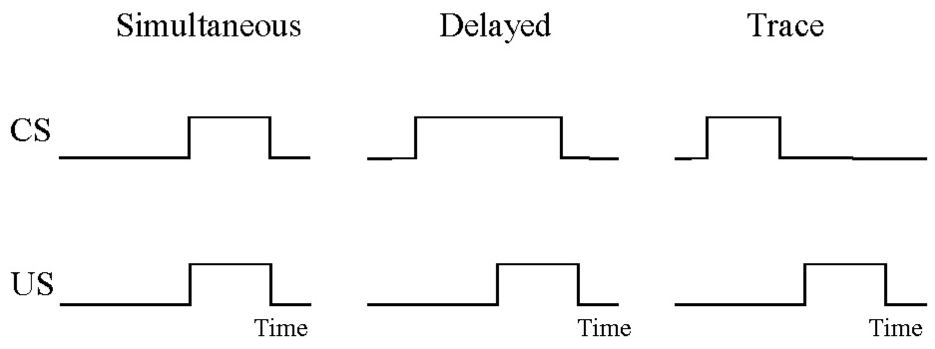

Figure 6.13: The three presentation methods for acquiring associations via classical conditioning.

There is, however, one very important issue that is highlighted when trying to change this typical experiment into one that can be done on the simulated DAC agent: the temporal aspect. In the standard experimental psychology point of view, the presentation of the two stimuli is considered one single event. However, in the simulation, we are immediately aware of the fact that these events must take place over time, and that the longer the stimuli are presented together, the more learning will occur. Note that this temporal issue is similar to the issue involving collisions discussed earlier (see section 5.2.2). Also, this can be compared to the one-sensor robot control system described earlier (see section 2.4.2), as it would seem that we would want sensors based on the change in the raw sensor value.

We have two ways of dealing with this in the simulation. We can vary the length of time that the stimuli are presented, or we can vary the learning rate. From the learning rule and experimentation, it is clear that these have equivalent effects on the behaviour of DAC learning.

The following diagrams show the activity of the three key values in this experiment, for the three different types of acquisition.

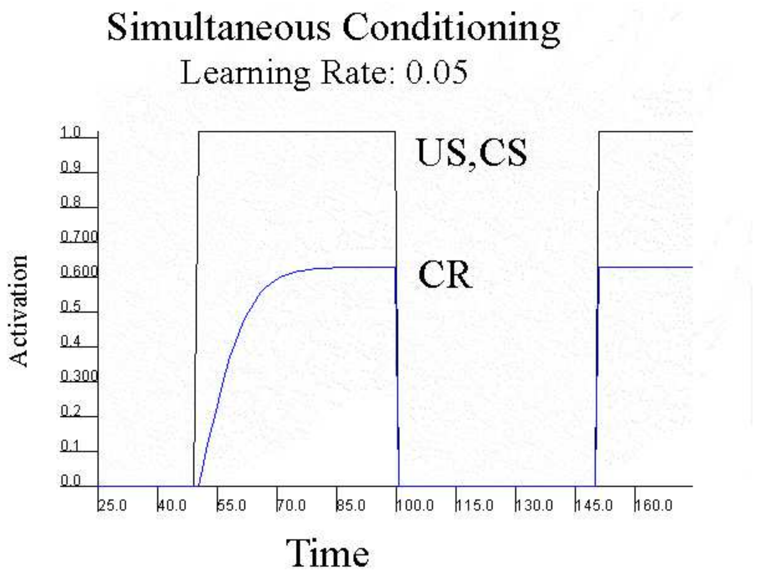

Figure 6.14: Results of simultaneous conditioning with DAC. Associations are successfully formed.

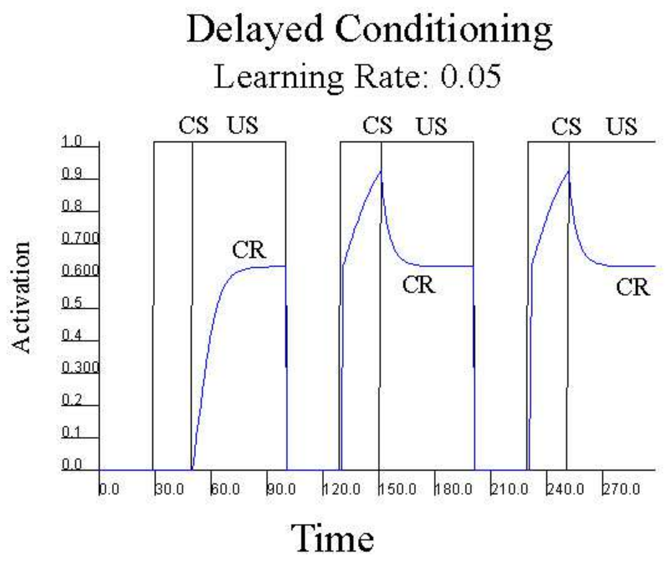

Figure 6.15: Results of delayed conditioning with DAC. Associations are formed, but with a non-typical pattern.

As can be seen from these results, DAC performs well on simultaneous and delayed conditioning, but completely fails on trace conditioning. Also, while delayed conditioning does result in a successfully formed association between the conditioned and unconditioned stimuli (as evidenced by the CR response before the unconditioned stimulus (US) is encountered), the pattern of activation is atypical. In particular, DAC actually gives a stronger response to the presence of the

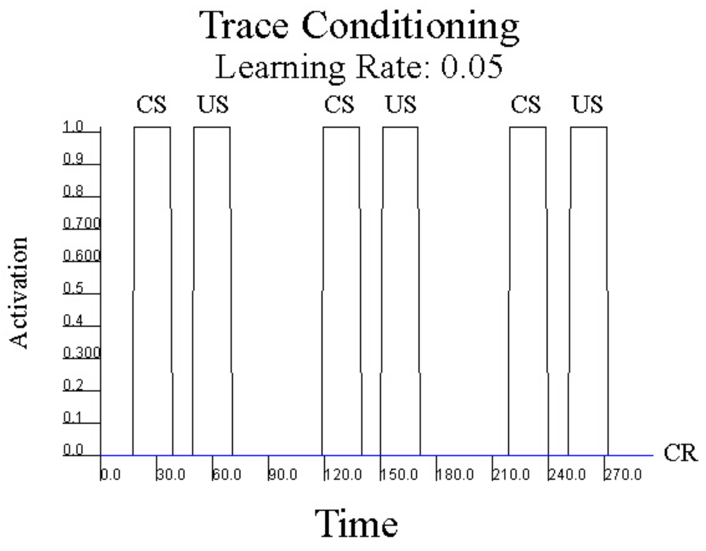

Figure 6.16: Results of trace conditioning with DAC. No associations are formed.

CS alone, as compared to the presence of both the CS and the US.

The fact that trace conditioning does not work is expected, since DAC has no internal temporal aspects. It was hoped that some evidence of trace conditioning could appear due to the temporal continuity of the agent's interaction with its environment. However, this particular test does not make use of environmental interaction in any way, and it proved difficult to come up with an effective alternate test for this given the limited sensor capabilities of the agent.

Also, it must be pointed out that the simultaneous and delayed conditioning trials exhibited a characteristic not common to natural classical conditioning. This is the afore-mentioned temporal aspect of a stimuli being spread over time. In both of these situations, the strength of the learned association is based on the amount of time the agent has been exposed to the relationship, not the number of times the relationship has been shown to it. For instance, an association can be fully formed by presenting the US and the CS together once for a long period of time, rather than presenting them for short periods of time over and over again. How long this period of time has to be can be tuned via the learning rate.

6.3.2 Extinction

The main point of extinction is that an agent must be able to lose an association that it has learned. That is, once an association is formed, if the agent makes a prediction based on that association and that prediction is wrong, then the strength of that association should be reduced.

There is also an interesting complication of this process called "spontaneous recovery", which is observed after an association is extinguished. If the creature is allowed a "rest time" (a period of time where there is no opportunity to test the association: i.e. neither the conditioned stimulus nor the unconditioned stimulus is presented), and is then presented with the conditioned stimulus, the association will occur (that is, the creature will perform the conditioned response).

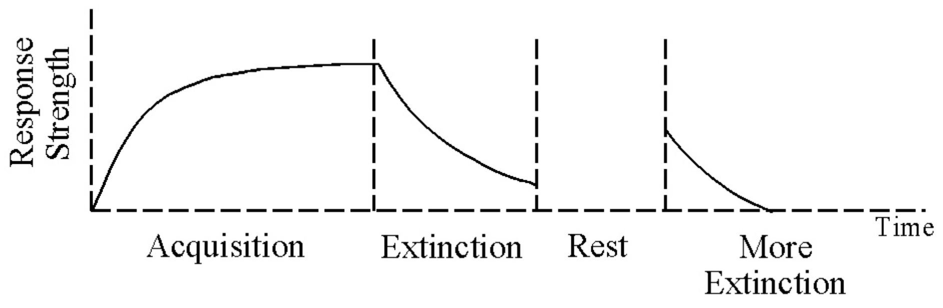

Figure 6.17: The typical classical conditioning extinction pattern.

To test DAC's capabilities in this regard, two experiments were done. In the first, the agent was initially trained using simultaneous conditioning as done in the previous section. This formed a simple association between one conditioned stimulus sensor and one unconditioned stimulus sensor. Then, the response of the agent was observed while presenting the conditioned stimulus multiple times without the unconditioned sensor ever being active. In the second experiment, the agent was initially trained using the wall-concept approach discussed earlier (see section 6.2), and the same extinguishing procedure was applied.

In both of these experiments, DAC failed. In neither situation did any extinguishing at all occur. The response to the conditioned stimulus stayed constant over the repeated tests. In hindsight, this is not surprising, since this is exactly what the DAC system does in its original form; the agent associated being close to walls with collisions, and then continues to use that association even though it does not actually collide with any walls any more.

The explanation as to why the DAC learning algorithm acts in this manner is reasonably clear from the learning rule itself. The weights are modified based on the difference between the current range sensor values and the predicted range sensor values based on the output of the collision sensors. However, that output from the collision sensors is the output after the collision sensors have been modified by the associations between the range sensors and the collision sensors. Thus, even if there is no physical collision, the learning algorithm works based on the predicted collision, and thus does not un-learn this association. It "hallucinates" the collisions, and then learns based on those hallucinations.

To clarify this explanation, consider the simple case of one conditioning sensor (the range sensor) connected to one unconditioned sensor (the concept sensor). Using the formulae for the DAC learning rule (see section 5.2.2), we find that the change in the connection strength between the neurons is:

```
dw=n*v*(u-e)
dw=n*v*(u-w*v)

v=w*u+c
```

- n is the learning rate
- v is the output of the concept sensor
- u is the range sensor
- c is the input to the concept sensor (the actual sensor value without the influence of the learned range sensor association)

Thus, the weight only stops changing when either v is zero, or when (u-w*v) is zero. Now, what happens during the extinction process? Since an association has been formed previously between the range and concept sensors, v will be non-zero due to the learned association. Thus, the learning will stop when (u-w*v) is zero.

```
  u-w*v
= u-w*(w*u+c)
= u-u*w*w-c*w
```

Since c=0 during the extinction process, this means that the value for w can end up at w=1, not at the expected w=0. In any case, from this simple example we can see that the DAC learning algorithm cannot support extinction.

Another analysis of this inability to extinguish would be to note that w=0 is an unstable equilibrium, so that even without the concept sensor (c) being active, the connection weight cannot settle down to w=0. To explain this, consider the case where c is always 0 (i.e. the concept sensor is always off) Now, the weight change is:

```
dw=n*(w*u+c)*(u-w*(w*u+c))
dw=n*(w*u)*(u-w*w*u)
dw=n*(u*u)*(w-w*w*w)
```

So, if w is exatly zero, it will stay at zero, since (w-w*w*w) will also be zero. Now, if, due to prior learning (or due to some random event), w is slightly positive, then (w-w*w*w) will also be positive. So, if u (the range sensor) is non-zero, then dw will also be positive. This then increases w, and the system enters a positive feedback loop which will eventually settle at w=1. Note that this process happens with c always being zero. Thus, DAC forms the association without the concept ever being encountered. This association will stay because w=1 is a stable equilibrium: a slight variation in w causes dw to have the opposite sign, forming a negative feedback loop which settles back to w=1. It should also be noted that this effect will happen even if there was a continuous decay term added that gradually reduced the strengths of the connection weights (unless, of course, that decay term is so strong as to disrupt all chance of learning).

Of course, this analysis only holds for c=0 and for systems with only one unconditioned sensor and one conditioning sensor. However, it seems clear that once an association is formed, DAC has a very hard time of getting rid of it. This means that DAC assumes that the environment it is inhabiting is static. Its functionality is based on the (generally false) idea that the associations it is forming will always be present. This is a severe limitation, and it is unclear if DAC can be modified to avoid this problem.

6.3.3 Generalization

Generalization describes classical conditioning's ability to handle novel situations. In particular, it is observed when a creature is conditioned with a particular stimulus, and is then confronted with a stimulus it has never before encountered, but is somewhat "similar" to the original stimulus. The result is that the agent will produce the conditioned response, but to a lesser degree. The more different the two stimuli, the weaker the response.

However, there is no obvious and clear way of measuring the "similarity" of any two stimuli. In conditioning experiments, "similarity" tends to mean things like tones of similar frequencies or objects of similar size. Thus, to perform these experiments on DAC, a modified version of the concept experiments were done.

The experiments were based on the corridor-concept experiments. The agent was first trained (as in the previous experiment) to recognize a corridor of a specific width (100). Then, the agent was forced to run through a corridor of a different width with learning turned off. By recording and averaging its prediction (the concept it was trained on), we can arrive at the agent's prediction for the new stimulus. Note that averaging the prediction was necessary, because the concept output varies significantly as the agent moves down the corridor.

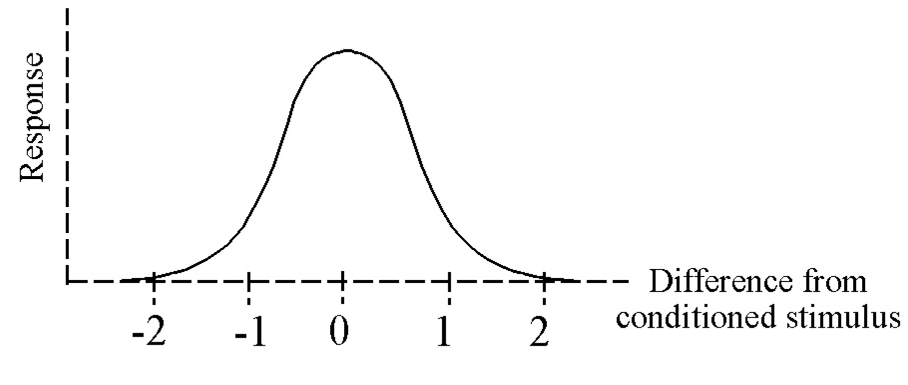

Figure 6.18: The typical generalization response.

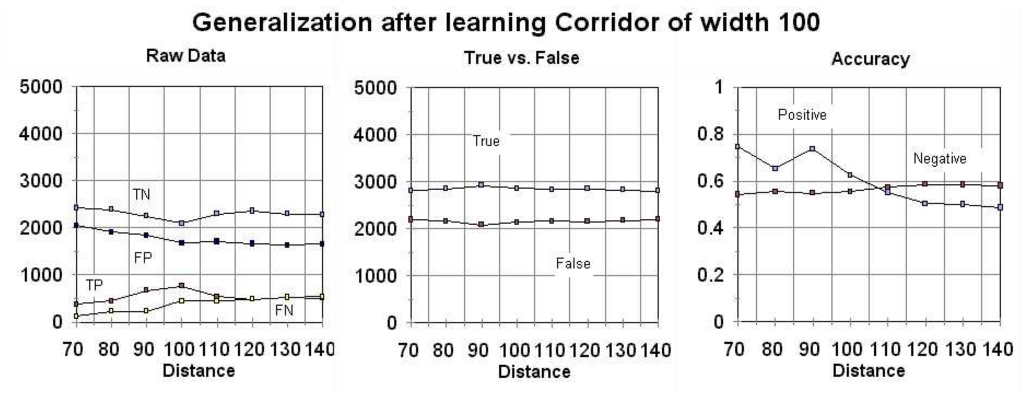

Figure 6.19: Results of generalization after a learning period on a corridor of width 100.

As can be seen, there is no clear bell-shaped generalization drop-off for this particular test. In fact, comparing the above graph to the graph for the corridor concept experiment (see section 6.2.1) reveals no significant differences at all. In other words, the system is not learning to identify corridors of various widths; it would be more accurate to say that the system is identifying corridors in general, and corridors of certain widths are more easily identifiable than others. Thus, the learning is generalized, but not in the manner expected.

It is certainly possible that other generalization tasks may be handled in a manner closer to the standards of classical conditioning. These other tests would perhaps require more sensors of different types and a more complex environment. However, failure on this simple test is not a promising beginning.

6.3.4 Specialization

Specialization is a process which occurs after generalization. When confronted initially with two similar stimuli, the agent associates both with the same conditioned response, due to generalization. If, however, further trials are done where the unconditioned stimulus is presented when only one of the two "similar" stimuli are presented, then the agent will eventually learn to distinguish between the two. It can learn to respond differently to the two stimuli, even though they were initially deemed to be "similar". We thus see that similarity is not only difficult to define, but is also dependent on the agent's experiential history.

Naturally, the experiments to test specialization are an extension of the generalization experiments. The exact process described above was done with the two stimuli being two corridors of different widths (70 and 140). The agent was first placed in an environment with a corridor of width 70, and whenever the agent was in the corridor, its concept sensor was activated. Then, it was placed in an environment with a width 140 corridor, and its concept sensor was never activated. The prediction from classical conditioning would be that the agent would eventually learn to distinguish between the two sizes of corridor, even though it failed to do so initially.

The result from this experiment is that DAC failed to learn in this manner, even after 100 repeated trials. Various other combinations of corridor widths were also tried, but with no success. The agent's predictions did not vary in a statistically significant manner.

This is, of course, unsurprising. Experiments described previously in this thesis (see section 6.3.2) have established that DAC does not exhibit extinction, and it is unclear how specialization could possibly work without extinction. Thus, this test did not provide any particularly new insight into DAC's functioning, other than arguing that its inability to lose old associations causes it to also be unable to fine-tune previously learned associations.

6.4 Discussion of Results

I think that it is clear from the results of the previous experiments that DAC is not a full model of classical conditioning. Of course, it was never supposed to be a full model, but merely a step in that direction. After going through the previous test suite of experiments, however, I feel that we have a much clearer conception of exactly which aspects of learning need to be the focus of future efforts.

The first major missing aspect in DAC is extinction. As we have seen, without extinction the agent cannot adapt to a changing environment (see section 6.3.2), nor can it adjust and fine-tune its learned associations to become more appropriate (see section 6.3.4). Adding this to DAC is non-trivial: it is not sufficient to merely implement a decay parameter that causes associations to disappear slowly over time. A true extinction system could perhaps be based on looking at when the predictions of the system are wrong, not purely based on how often they are used. Such a model may even use a learned inhibition to handle the extinction, and then a decay in that inhibition could explain spontaneous recovery. How such a system would interact with the concept-formation characteristics embedded in this view of classical condtioning is unknown.

The second major issue is the limitations in the sorts of "concepts" that can be formed. We have seen how DAC is only capable of forming associations between fairly simple patterns, and that it has severe difficulties in forming associations based on negative correlations (see section 6.2.1). To combat this problem, the reader is directed to the first half of this thesis, which presents many approaches which seem suitable for finding more complex patterns. In particular, the sparse representation networks (see section 3.4.4) may be a richer source for forming concepts.

A further step to expanding the concept-formation ability of this system could be surprisingly simple: add more sensors. With more sensors of different types, the system will suddenly have a much broader range of data with which associations can be formed. This approach may be compared to TNGS's reliance on huge numbers of randomly generated data pre-processors (see section 3.3.3).

It is also hoped that this series of experiments has some influence on our ideas about classical conditioning. In particular, we have seen some conflict around defining what "presenting a stimulus" means. Does it mean presenting it for one instant, or over a period of time? How long a period of time? Importantly, how do these alternate presentations affect the learning? With DAC, these options significantly change the outcome of the experiments. It is not clear how much this is the case for natural creatures.

Also, there has been an underlying theme in this paper about combining concept formation in with conditioning. We have seen that these two aspects of learning may be tightly related, and are, at the very least, dependent on one another. What use is conditioning without found regularities in the environment to do the conditioning on? And what use is concept formation if you are not doing anything with those concepts? I do not see how these two issues can be investigated separately, at least not at this low level of simple behaviours that likely underlie all learning.

Bibliography

[1] Ashby, W. R. An Introduction to Cybernetics. London: Methuen. 1956.

[2] Atkinson, R. L., and Hilgard, E. R. (eds) Hilgard's Introduction to Psychology. 12th ed. San Diego: Harcourt Brace. 1996.

[3] Balkenius, C. Natural Intelligence in Artificial Creatures. Lund University Cognitive Studies. 1995

[4] Beer, R. "Framing the Debate Between Computational and Dynamical Approaches to Cognitive Science." Behavioural and Brain Sciences 21 (5): 630. 1998.

[5] Braitenberg, V. Vehicles: Experiments in Synthetic Psychology. MIT Press, 1984

[6] Cruse, H. "Coordination of leg movement in walking animals." From Animals to Animats 1: Proceedings of the First International Conference on the Simulation of Adaptive Behavior. Cambridge, MA: MIT Press. 105-119. 1991.

[7] Dennis, W. "Causes of retardation among institutional children." Journal of Genetic Psychology 96: 17-59. 1960.

[8] Dorigo, M. and Colombetti, M. Robot Shaping: An Experiment in Behavior Engineering. Cambridge, MA: MIT Press. 1997.

[9] Edelman, G. E. Neural Darwinism: The theory of neuronal group selection. New York: Basic Books. 1987

[10] Fawcett, T. "Feature Discovery for Inductive Concept Learning." Technical Report UM-CS-1900-015, Department of Computer Science, University of Massachusetts. 1990.

[11] Garner, M. "Mathematical Games." Scientific American 223 (4): 120-123. 1970.

[12] Harpur, G. Low Entropy Coding with Unsupervised Neural Networks. PhD Thesis, University of Cambridge. 1997.

[13] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. "The wake-sleep algorithm for unsupervised neural networks." Science 268: 1158-1161. 1995.

[14] Hinton, G.E., and Ghahramani, Z. "Generative Models for Discovering Sparse Distributed Representations." Philosophical Transactions of the Royal Society, B, 352: 17-1190. 1997.

[15] id Software. "Quake 2." Available at http://www.idsoftware.com/quake2/. Accessed July 2000.

[16] Jakobi, N. "The Minimal Simulation Approach To Evolutionary Robotics," in Proceedings of ER'98, T. Gomi (ed.), AI Systems Books. 1998.

[17] Jakobi, N. Minimal Simulations For Evolutionary Robotics. DPhil Thesis, School of Cognitive and Computing Sciences, University of Sussex. 1998.

[18] Karmiloff-Smith, A. Beyond Modularity: A Developmental Perspective on Cognitive Science. Cambridge, MA: MIT Press. 1992.

[19] Keijzer, F. A. "Some Armchair Worries about Wheeled Behavior." From Animals to Animats 5: Proceedings of the Fifth International Conference on the Simulation of Adaptive Behavior. Cambridge, MA: MIT Press. 1998.

[20] Maris, M. "One Sensor Robot Control." Technical Report 94-08, Artificial Intelligence Laboratory, University of Zurich. 1994.

[21] Mataric, M. and Cliff, D. "Challenges in Evolving Controllers for Physical Robots." Evolutional Robotics: Robotics and Autonomous Systems 19(1): 67-83. 1996.

[22] Nolfi, S. "Evolving non-trivial behavior on autonomour robots: Adaption is more powerful than decomposition and integration." Technical Report, Institute of Psychology, CNR, Rome. 1997.

[23] Pederson, B. Q2Java. Available at http://www.planetquake.com/q2java/. Accessed July 2000.

[24] Pfeifer, R. "Building 'Fungus Eaters': Design Principles of Autonomous Agents." From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press. 1996.

[25] Quick, T., Dautenhahn, K., Nehaniv, C., and Roberts, G. "The Essence of Embodiment: A Framework for Understanding and Exploiting Structural Coupling Between System and Environment." Proceedings of the Third International Conference on Computing Anticipatory Systems. Liege, Belgium. 1999.

[26] Rescorla, R. A., and Wagner, A. R. "A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement." In A. H. Black and W. F. Prokasy (Eds), Classical Conditioning II: Current research and theory (pp. 64-99). New York: Appleton-Century-Crofts. 1972.

[27] Reynolds, C. W. "Flocks, Herds, and Schools: A Distributed Behavioral Model." Computer Graphics 21(4): 25-34. 1987.

[28] Rucci et al. “A robotic system emulating the adaptive orienting behavior of the barn owl.” Proceedings of the 1997 IEEE International Conference on Robotics and Automation. 1997.

[29] Rutkowska, J. “Reassessing Piaget’s Theory of Sensorimotor Intelligence: A View from Cognitive Science.” Cognitive Science Research Paper 369. University of Sussex. 1995

[30] Rutkowska, J. “Computation, Dynamics, and Sensory-Motor Development.” Cognitive Science Research Paper 460. University of Sussex. 1996.

[31] Rutkowska, J. “What’s Value Worth? Constraining Unsupervised Behaviour Acquisition.” Fourth European Conference on Artificial Life, Husbands P. and Harvey I. (eds), MIT Press. 1997.

[32] Sutton, R. “Dyna, an Integrated Architecture for Learning, Planning, and Reacting.” Working Notes of the 1991 AAAI Spring Symposium: 151-155. 1991.

[33] Svedsen, B. (Decker). “QuArK: Quake Army Knife.” Available at http://www.planetquake.com/quark/. Accessed July, 2000.

[34] Thelen, E. and Smith, L. A Dynamic Systems Approach to the Development of Cognition and Action. Cambridge, MA: MIT Press. 1994.

[35] Thornton, C. “Brave Mobots use Representation: Emergence of Representation in Fight-or-Flight Learning.” Cognitive Science Research Paper 401. University of Sussex. 1997.

[36] Tsodyks M, Pawelzik K, and Markram H. Neural networks with dynamic synapses. Neural Computation 10, 821-835. 1998.

[37] van Gelder, T. “The Dynamical Hypothesis in Cognitive Science.” Behavioural and Brain Sciences 21:1-14. 1998.

[38] Veogtlin, T. and Verschure, P.F.M.J. “What can robots tell us about brains? A synthetic approach towards the study of learning and problem solving,” Reviews in the Neurosciences, 10 (3-4): 291-310. 1999.

[39] Verschure, P. and Coolen, A. “Adaptive Fields: Distributed representations of classically conditioned associations. Network 2: 189-206. 1991.

[40] Verschure, P. and Pfeifer, R. “Categorization, Representations, and the Dynamics of System-Environment Interaction: a case study in autonomous systems.” Animals to Animats 2: 210-217. Cambridge, MA: MIT Press. 1992

[41] Verschure et al. “Distributed adaptive control: the self-organization of structured behavior,” Robotics and Autonomous Systems 9: 181-196. 1992.

[42] Verschure, P. “Connectionist explanation: taking positions in the mind-brain dilemma.” Neural networks and a New AI, G. Dorffner (ed.):125-181. London: Chapman & Hall. 1994.

[43] Verschure, P. and Voegtlin, T. “A bottom up approach towards the acquisition and expression of sequential representations applied to a behaving real-world device: Distributed Adaptive Control III.” Neural Networks 11:1531-1549. 1999.

[44] Walter, W. G. “An Imitation of Life.” Scientific American: 42-45. May 1950.

[45] Watt, J. “Sculpture and Jewellery Index: Quake Home-Brew.” Available at http://www.kinetic-arts.demon.co.uk/. Accessed July, 2000.

[46] Wojtowicz, M. Mirek’s Cellebration. Freeware software. Available at http://www.mirwoj.opus.chelm.pl. Accessed December, 1999.

[47] Yamauchi, B. and Beer, R. “Sequential Behavior and Learning in Evolved Dynamical Neural Networks.” Adaptive Behavior 2(3): 219-246. 1993.

[48] Yamauchi, B. and Beer, R. “Spatial Learning for Navigation in Dynamic Environments.” IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 26(3): 496-505. 1996.