Linköping Studies in Science and Technology
Thesis No. 397

Behavior Representation by Growing a Learning Tree

Tomas Landelius

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1993

Abstract

The work presented in this thesis is based on the basic idea of learning by reinforcement, within the theory of behaviorism. The reason for this choice is the generality of such an approach, especially that the reinforcement learning paradigm allows systems to be designed which can improve their behavior beyond that of their teacher. The role of the teacher is to define the reinforcement function, which acts as a description of the problem the machine is to solve.

Learning is considered to be a bootstrapping procedure. Fragmented past experience, of what to do when performing well, is used for response generation. The new response, in its turn, adds more information to the system about the environment. Gained knowledge is represented by a behavior probability density function. This density function is approximated with a number of normal distributions which are stored in the nodes of a binary tree. The tree structure is grown by applying a recursive algorithm to the stored stimuli-response combinations, called *decisions*. By considering both the response *and* the stimulus, the system is able to bring meaning to structures in the input signal. The recursive algorithm is first applied to the whole set of stored decisions. A mean decision vector and a covariance matrix are calculated and stored in the root node. The decision space is then partitioned into two halves across the direction of maximal data variation. This procedure is now repeated recursively for each of the two halves of the decision space, forming a binary tree with mean vectors and covariance matrices in its nodes.

The tree is the system's guide to response generation. Given a stimulus, the system searches for responses likely to result in highly reinforced decisions. This is accomplished by treating the sum of the normal distributions in the leaves as distribution describing the behavior of the system. The sum of normal distributions, with the current stimulus held fixed, is finally used for random generation of the response.

This procedure makes it possible for the system to have several equally plausible responses to one stimulus. Not applying maximum likelihood principles will make the system more explorative and reduce its risk of being trapped in local minima.

The performance and complexity of the learning tree is investigated and compared to some well known alternative methods. Presented are also some simple, yet principally important, experiments verifying the behavior of the proposed algorithm.

Acknowledgments

Writing a thesis is a time consuming process. Luckily, the taste of time was not too pleasant this summer. Thanks to the dreadful result produced by this summers clerk of the weather, my indoor typing activity became a lot more bearable than first feared.

I am by no means the only one to take credit for the work that led up to the results presented here. There are some people who I owe a special debt of gratitude. Without them, this thesis had never been born.

First of all I wish to thank the members of the Computer Vision Laboratory for many a valuable discussion, and for encouraging my efforts by providing me with a stimulating research environment.

I would also like to thank professor Gösta Granlund for introducing me to the field of computer vision and for pointing out the close relationship between acting, seeing and learning.

Many thanks are expressed to my supervisor, associate professor Hans Knutsson, for sharing his idea of a learning tree with me, and for his never ending stream of suggestions and ideas, which sometimes makes my train of thoughts run off the rails.

Special thanks to Magnus Borga and Klas Nordberg for spending some of their time bandying ideas with me. Particularly to Magnus for feeding back constructive critique on early and late versions of the manuscript, as well as proofreading it. Remaining errors are to be blamed on me, due to my last minute changes.

Finally, I would like to thank my parents for their constant encouragement and last but not least I would like to express a very special gratitude to my wife for life, Rebecca. Thank you for love, support, and wholesome distractions.

This research was sponsored by NUTEK, the Swedish National Board for Industrial and Technical Development, which is gratefully acknowledged.

About the Cover

Throughout history, learning and knowledge have often been symbolized with the fruit of the apple tree. The cover of this thesis is no exception. The picture of this tree was originally cut out in wood and printed on an unpublished letter from a Swedish history writer named Olof Rudbeck, to the Chancellor of the Swedish Universities, in 1674. This letter can be found in the collections of the National Archives. At the time when this letter was written, Sweden was one of the great powers in Europe. Rudbeck put a lot of effort in rewriting history, locating all major events and persons to the northern countries, especially Sweden. The classic gods were maybe not too pleased to hear some of his statements. In his book Atlantica, he tells the true story about the "golden apples" of Minerva. The apples, he claims, were originally to be found in the northern mythology as the apples of Iduna, which is the goddess of wisdom in northern mythology. Hence, it is only fallen fruit which the people of the south brought back home from the northern countries.

Contents

Chapter 1

Introduction

This work deals with issues concerning learning and knowledge related to memory and machines. The strong connection between learning, knowledge and memory is found in most explanations of what it means to learn something. As an example, consider this paragraph about knowing (Wittgenstein, 1958):

> *150. The grammar of the word "knows" is evidently closely related to that of "can", "is able to". But also closely related to that of "understands". ('Mastery of a technique',)*

Now, since learning could be described as e.g. to master through experience, to gain knowledge or to memorize, it becomes clear that all these concepts are intimately related. The engaging question of how something is learned has divided philosophers into two opposing parties, the empiricists and the rationalists. This matter will be discussed in more detail in chapter 2.

The apple has been a symbol for knowledge for quite some time. Olof Rudbeck, who was mentioned in the section about the cover, is certainly not the only one who could be condemned for hubris when dealing with the fruit of the apple tree. Maybe apples were the forbidden fruit that Adam and Eve ate, from the tree of knowledge, committing the very first act of hubris in history. The apple also acts as a symbol of power. As such, it first appeared with the Roman emperors symbolizing world dominion. This tradition is still alive in e.g. Sweden where a golden apple is one of the regalia. An old Greek word for power, or to make something possible, is *mēcháně*. This word is related to another Greek word, *machina*, which is the origin of the word machine. The quest for machines that learn from memorized experiences is one the great challenges of modern science. Machines, which today are claimed to possess an ability to learn show a remarkable rigid behavior, far behind that of the simplest biological organisms. However, a machine *capable* of learning would indeed possess the attributes here ascribed to the word machine. It would by all means be a very powerful tool, making things possible which are out of reach for any human programming effort. That there exists limits for human programming abilities will be made obvious in the next section.

1.1 Background

One of the most important ventures in modern society is the development of industrial robots. However, the range of tasks in which such machines can be used is presently limited due to their inability to learn. Present "intelligent" machines are more or less rigid. The responses of the system are pre-specified, by human programmers, for all foreseeable situations. It takes little thought to realize that very few real life tasks can be solved by this strategy.

Consider e.g. the so called *expert systems*. These are systems with much knowledge but very limited inference capabilities. It has been suggested (Reddy, 1988) that at least $70 \cdot 10^3$ pieces of information are needed to come anywhere near the performance of human experts. Building such a system and feeding it with the required amount of information will call for an enormous programming effort. Taking the reliability of the produced software into account, very large programmed systems becomes infeasible. It is known that over 60% of the bugs in the software can be traced back to activities before the writing of the code. Humans are human at making errors but poor at removing them. Also, even efficient software inspection methods will not remove more than about 80% of the bugs. All errors are not found and corrections may give raise to new ones. Thus, machines with something like human performance, in non expert domains, seems impossible to achieve by programming.

Instead, the wish must be for systems which to some extent are programmed, but where most of their emergent behavior is left to learning. This will produce robust and flexible machines which are able to cope with variations both in the environment and in the system itself. Now, if $70 \cdot 10^3$ memories are needed for a machine with domain specific knowledge, how many experiences will than a general purpose system need to acquire a human like behavior? The *million memory goal* is a speculative estimate of an upper bound on the memory capacity needed for human like performance (Omohundro, 1987). To get a feeling for this issue, consider the total possible amount of information input to the nervous system during a lifetime. What has been experienced after thirty years of life, containing about 10^8 seconds? Some 10^7 afferent fibers, each with a bandwidth of about 100 bits/s, have been feeding input to the brain. This results in the total amount of information being somewhere around 10^{18} bits. Now if a person, while awake, is considered to store a memory every 10 seconds some fifty million memories will be stored. A number of indicators suggests that the actual number of memories needed in various human tasks is of the order of a million. The number of words a typical individual uses in everyday reading and writing is about 10^4. According to Zipf's law their probability distribution is closely approximated with $0.1/i$, where i is the index to the i:th most probable word. From this it can be seen that the number of objects a person can identify, with a word or two, should be of the order of a million. In the game of "twenty-questions", a good player is able to identify an arbitrary object, i.e. to discriminate between $2^{20} \approx 10^6$ possibilities. Furthermore it seems reasonable to believe that a person learns no more than 100 new items a day during the 10000 days into adulthood. Another observation along this line of reasoning is that many

of the parallel nerve bundles from one brain area to another contain on the order of a million fibers.

Two of the main approaches in search of intelligent behavior are based on simulation of biological systems which possess this desired quality. One course is that of *artificial intelligence* (AI), which make use of introspection to find out how humans process information in an intelligent way. Then this behavior, or thought process, is imitated in a computer simulation. As a crude metaphor consider an approach to construct airplanes that flap their wings based on the inspection of birds who are known to be able to fly. In the same way as it is possible to build machines that fly without the physiology of birds, it should be possible to design machines that show an intelligent behavior but lack the thought process described by human psychology. The functionality of nature may be copied but need not be implemented literally.

The same thing can be said about the path in the opposite direction. This approach called *connectionism* states that the lack of intelligent behavior in learning machines of today is due to the difference in he underlying computing structure between present computers and biological systems. Intelligent behavior will emerge if the underlying computational structure is the right one. Now, the Church-Turing thesis denies the importance of any such difference (Minsky, 1967). It claims that a Turing machine, which in principle can be implemented on a serial computer of today, is capable of simulating any physical component system to arbitrary precision, deterministic as well as stochastic.

The route towards intelligent behavior seems to go through the implementation of good algorithms rather than through literate simulation of biological processes. A key feature of any good algorithm for machine learning is generalization. Learning systems must be able to build up most of their knowledge from experiences and then generalize this information to new situations. Choosing one generalization instead of another is to introduce bias. For a general purpose system it is impossible to determine how to bias its behavior towards important properties of a problem, and avoid minor details. However, geometric domains unlike the symbolic ones studied in AI, incorporate a natural bias, that of *continuity*. The *neural network* structures, suggested by the connectionists, work in a geometric domain and make use of this bias to perform a simple form of generalization, that of interpolation. One of their major drawbacks is the lack of techniques for designing neural nets given an engineering task to solve. Completely connected nets are able to implement any functional mapping from stimulus to response but grow to infeasible sizes when harder real life problems are faced. There is a tremendous pressure on the nervous system for representation, computation, and communication. Intelligent systems must be capable to learn efficient representations themselves. Also, they must be able to use these for response generation in order to improve both their performance and the representations. One solution to this problem is to introduce the bias of *locality*. If the computations in the system are local it is possible to apply focus of attention strategies and do serial processing. A single piece of high-level hardware may be used serially on different parts of the parallel stimuli depending on the context in which the system is currently operating.

An alternative to the methods described before, both of which are introspective

either with respect to human psychology or physiology, is the behavioristic approach. With this view as little as possible is assumed about the inside of the system and the interest is focused on the *closed loop* of stimuli and responses to and from the system. Analysis of stimuli which does not show up as responses is considered to be meaningless. The system presented in this thesis is based on a behavioristic view. It makes use of the two priors of continuity and locality, which should be harmless to the generality of the system. Some *functional* similarities, between biological systems and the algorithm proposed in this thesis, may deserve to be pointed out. Locality is found to be important to biological systems for applying serial processing to a massive amount of parallel data. Particularly the locus of memory and computation, appear to coincide in biological nervous systems (Omohundro, 1990). In the representation to be presented later, models of behavior and the experiences supporting them can be thought of as stored in the same place. Also the benefits of locality for focus of attention exploited in biological systems have a counterpart in so called branch and bound methods used in the suggested algorithm.

A binary tree is proposed as a structure for representation of system behavior. The global model of system behavior is fragmented and localized to a number of models, or modules, in the leaves of the tree. An interesting question is how many models that may be needed for real life tasks? Look at the human brain for another speculative estimate (Omohundro, 1987). From what is known, it seems as if the brain is made up of a number of functionally different areas joined together with ordered fiber bundles. Recent findings estimate the number of such modules to be about 200. Interesting human computations, such as visual recognition, takes times around 0.5 seconds to perform. Since the time intervals in the firing of an individual neuron are about 0.005 seconds, it seems to be at most 100 layers of neurons from stimulus to response. The function of a single module could be implemented with a few layers of neurons which means that probably some 20 to 40 modules are active along any path from input to output. The depth of a binary tree representing somewhere around 200 models will be about 8 which is of the same order of magnitude as the result from the above consideration. The functionality of the brain, with its 10^{11} neurons, is largely unknown. Hence, the estimates previously presented should be accepted with a grain of salt. Nevertheless they provide an indication of the bounds within which an intelligent system could be designed.

1.2 Organization of the Thesis

The basic idea of learning by reinforcement and its connection to the theory of behaviorism is discussed in chapter 2. Treated are also the issues concerning task definition in terms of reinforcements, and the question why stimuli and responses need to be handled together in order to make a useful description of the system's behavior.

Chapter 3 deals with the question of how to represent the behavior of the system. Representations using standard neural networks are compared to look-up structures, both with respect to structure and performance. It is concluded that the most flexi-

ble representation is the probability density function which describes the distribution of the experienced stimulus-response pairs. A tree like look-up structure is found to be useful for embedding the model of this probability distribution.

A presentation of the approach to approximate the probability density function with a number of normal distributions, located in the tree nodes, is made in chapter 4. The procedure of growing and using the tree in a bootstrapping procedure is described. Different performance issues and estimates are discussed and a number of simple experiments are conducted to illustrate some important properties of the algorithm. Parts of chapter 4 was presented at the conference on Adaptive and Learning Systems II, as part of SPIE's 1993 Symposium on Optical Engineering and Photonics in Aerospace and Remote Sensing in Orlando Florida, April 1993 (Landelius and Knutsson, 1993).

Finally the thesis is summed up in chapter 5 which also contains an outlook on important future research activities.

Chapter 2

Learning Theory

This chapter will give a rough description of *empiricism* and *rationalism*, the two main theories of learning, and describe in more detail two issues of interest to the learning algorithm described later in chapter 4 namely behaviorism and learning with reinforcement. There is a strong connection between learning and knowing. The theory of knowledge is studied by philosophers under the name of *epistemology*. Learning can be given a lax definition as "acquisition of knowledge through experience". Acquisition implies a change in a system's possession of knowledge before and after something is learned. A more rigid definition of learning is found in the standard work on learning theory, *Theories of learning* by Bower and Hilgard, from which much of the material in this chapter is borrowed (Bower and Hilgard, 1981):

> *Learning refers to the change in a subject's behavior or behavior potential to a given situation brought about by the subject's repeated experiences in that situation, provided that the behavior change cannot be explained on the basis of the subject's native response tendencies, maturation, or temporary states.*

In order not to have the terms system and subject mixed up an explanation may be called for. Subjects refer to biological systems and artificial systems refer to man made systems whereas the term system is used as a broader concept covering all sorts of systems.

2.1 Behaviorism

What causes the acquisition or the change of a system's behavior? In the answers to that question the concept of mind turns up. One should be a bit careful when speaking about minds and artificial systems since the answers where developed with humans and sometimes animals in mind. However, a review of two opposing answers may be worthwhile. There exists two theories in the philosophical community concerning how knowledge arise and the relation between experience and organization of mind. On one side are the empiricists and on the other the rationalists. Both admit that experience plays a role in acquiring knowledge but they differ in how

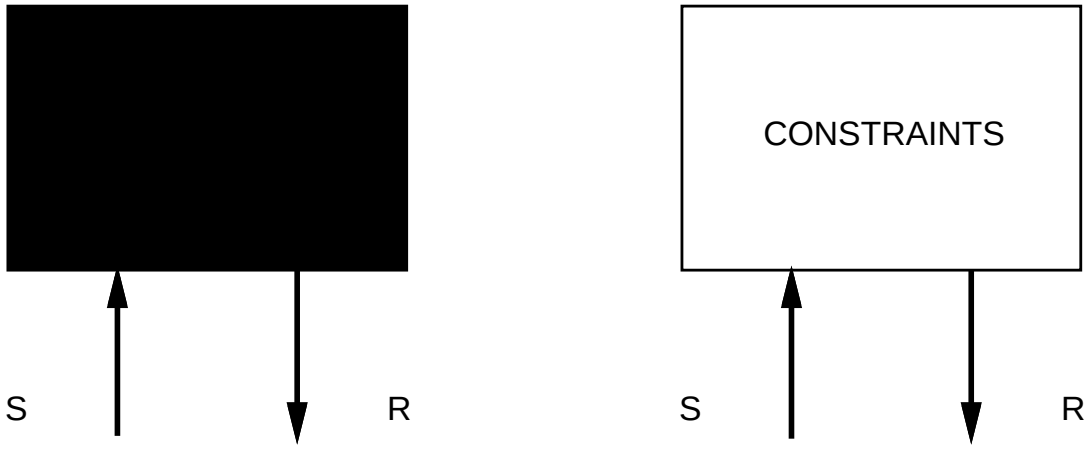

Figure 2.1: System views. Empiricism (left) and rationalism (right).

big a role that is. Later all behavioristic theories will be seen to share the view of empiricism.

Empiricism Empiricists claim that experience is the only source of knowledge. Experience in terms of sensory information is given a special emphasis. These sensory sensations become associated in the mind because they occur closely together in time or space as we interact with the environment. The degree of association is determined by how vivid, frequent, durative and recent the sensations are. Another characteristic for this view is that complex ideas are built from a stock of simpler ideas and the other way round implying that complex ideas can be reduced into simpler ones. Two learning mechanisms are recognized. First, sensory information is represented internally in terms of a *memory* containing the stimuli. Second, complex ideas are formed by connecting, or *associating*, simple ideas that are experienced contiguously, i.e. close together in space or time. These associations are then used for prediction and explanation of events. The possibility of reflection i.e. to recall memories, compare them and store the conclusion as a new association is what makes the difference between a system based on empiricism and a passive stimuli recorder. The strong position of association in empiricism is well illustrated with a quote from Pavlov: "The same thing can be said of our thinking. Beyond association there is nothing more in it" (Pavlov, 1955). Early promoters of the view of empiricism where the philosophers John Locke, David Hume and John Stuart Mill.

Rationalism Rationalists such as Descartes, Leibnitz and Kant on the other hand, claim that reason, not experience, is the prime source of knowledge. Kant put it this way (Kant, 1952): "Although all our knowledge begins with experience, it by no means follows that it all originates from experience." Sensory data is considered as raw material which needs to be interpreted according to certain innate assumptions or self evident truths with which the mind is constrained when it all begins. One example of such a constraint is the fact that all events take place in a spatio-temporal

framework. Kant saw this as the projection of the self evident truth of Euclidean geometry, with which all of us are born. For real knowledge it is necessary to constrain the thought relationships over and above the raw sensory data. What answer does a rationalist give to the question about how these constraints was acquired? One is natural selection through evolution, and another that he does not know how they evolved but that he knows that they are present in systems capable of learning. Evolution differs from learning according to the rationalists since the forms of learning under their consideration are biased by innate perceptual assumptions without which learning as they see it wouldn't take place.

To make a distinction between learning and evolution seems quite hard to motivate, at least if considering the more lax definition of learning mentioned earlier. Hence evolution could also be seen as a form of learning since at another higher level of abstraction spices gain knowledge through experiences in the environment. Most of the common principles referred to by the rationalists are found by introspection and may not be relevant when applied to non human learning systems. Using too specific metaphors when trying to imitate biological phenomena may be misleading as demonstrated with the bird-airplane metaphor in chapter 1. Also principles guiding human learning found by introspection have been gained through evolution and may not be similar to the principles guiding learning at the level of evolution. However, if common principles shared by all learning systems exist they are still to be looked for. When designing machine learning systems not to imitate the human thought process in terms of logic and language but to perform problem solving in terms of response generation it seems as if the view of empiricism is the one to adept. This does not mean that one has to deny the existence of common principles to learning systems since such principles could be imposed on the system if and when discovered. In absence of constraints to put on the system one has to look at what can be done with as few assumptions as possible about the "inside" of the system, see fig. 2.1. This ends up in a theory much in line with that of Skinner who claimed to have no theory at all (Skinner, 1950).

The view of empiricism is shared by all behavioristic theories. Watson, the leader of the behavioristic revolution explains learning as Pavlonian conditioning. A response is associated with a stimulus such that after the learning trial, the stimulus will result in the associated response with a certain probability. The learning system can be viewed as a black box which behavior is modeled only with a probability distribution, $p(\mathbf{x})$, over the product space D of the stimuli and response spaces, $\mathbf{x} \in D = S \times R$. Hence a stimulus and its associated response can be treated together as a *decision*. When the system has learned to solve a problem its decisions will be generated from an *optimal behavior distribution, OBD*. In fig. 2.2 the optimal system behavior is to produce a sine wave. This behavior can be described with a probability density function having its maxima along the decisions which constitute the optimal behavior. Probabilities are reflected in the grey levels of the graph at the bottom of fig. 2.2, the whiter the more probable a decision. Learning from the system's point of view is then equal to estimating the OBD. Since nothing is known by the system a priori it will have to start with what Locke called a *tabula rasa*, an

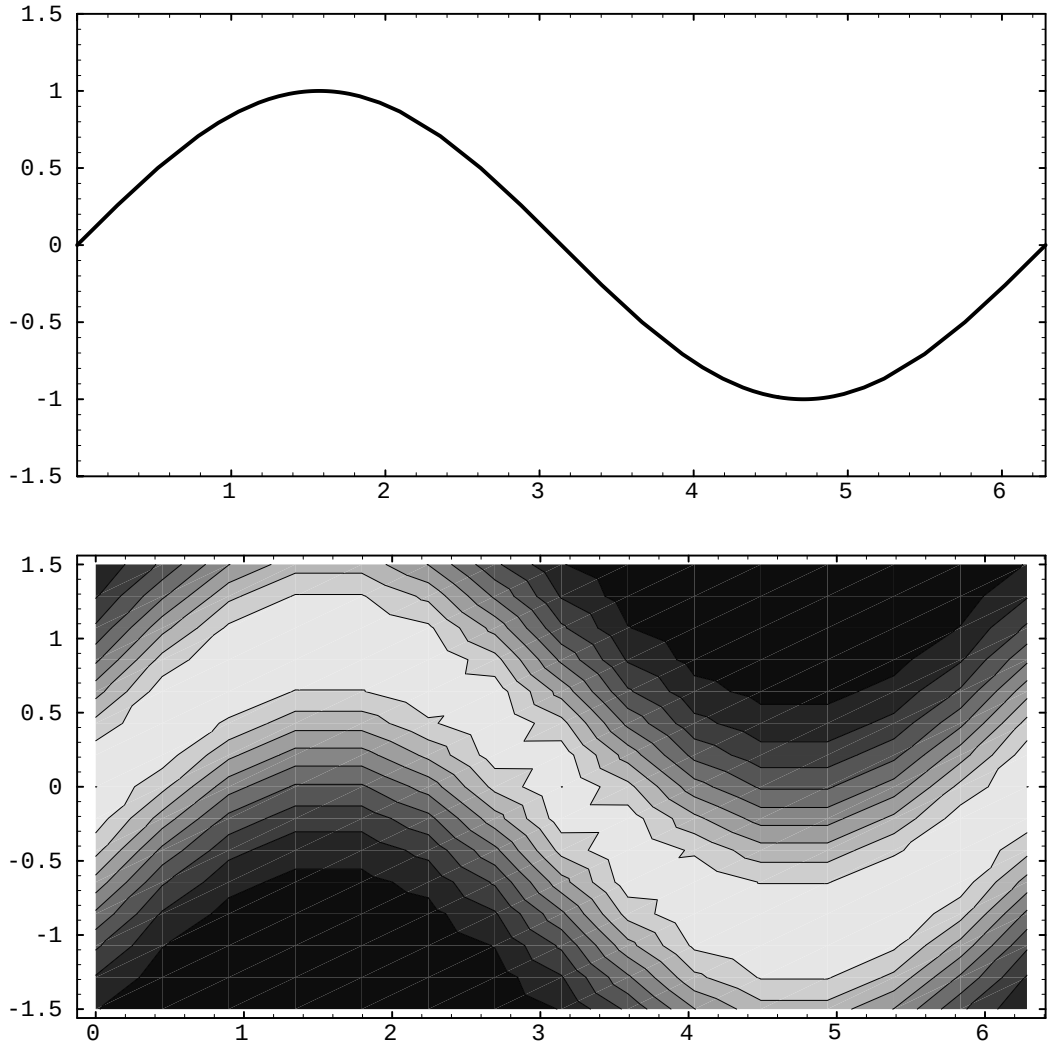

Figure 2.2: Representing an optimal behavior (top) with a distribution (bottom).

unused blackboard. This presumption is similar to the one made in the branch of statistics studying model-free estimation, where no parametric models are assumed. The estimation error in model-free estimation can be divided into two terms, bias and variance (Geman et al., 1992). The bias error is related to the system bias discussed earlier. A system biased towards a certain behavior from which it is able to depart only to a certain extent, will mainly suffer from an error in bias. A more flexible system will be able to reduce the error in bias but will instead pay for its flexibility with a large error due to variance. This is why flexible systems need large training data sets to reduce their error. Learning of complex tasks is claimed by Geman to be essentially impossible without carefully introducing system bias, e.g. by an appropriate choice of stimuli representation. Again this claim must be rejected if evolution is seen as a form of learning since a good representation of stimuli also must be learned by the system.

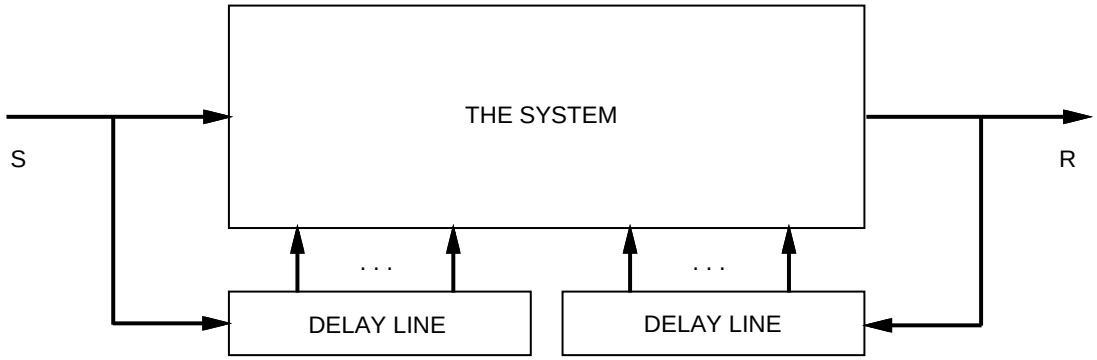

Figure 2.3: Incorporating system memory into the framework.

The notion of stimuli should not be taken only to mean current sensory input. A key mechanism for learning according to the empiricists is the system's ability to keep a memory of both past stimuli and responses. Such a mechanism can easily be incorporated into the framework of probability density estimation, by presenting the system with any portion of previous stimuli and responses at each time step, (Narendra and Parthasarathy, 1990) see fig. 2.3. The amount of old information that the system can use is determined by the length of the delay lines. This structure reduces to the well known IIR filter if the system is only able to produce weighted linear sums of its inputs.

Now having decided to work with a behavioristic model will not be enough. How is the system to know what to do? In the following section a number of different views on this topic within the behavioristic theory are presented and evaluated.

2.2 Learning Methodologies

The reason for building a learning system must be that there exist one or more tasks that the system is expected to learn how to perform. What kind of tasks are there then? Barto makes the distinction between *nonassociative* and *associative* learning tasks (Barto, 1992). By the way, this is a reference that contains a lot of references to relevant literature on reinforcement learning, a matter to be discussed in the next section. Nonassociative tasks calls for the system to find an optimal response without the use of stimuli information, see the middle of fig. 2.4. This type of systems has been investigated under the name of *learning automata* by e.g. (Narendra and Thathachar, 1974). In associative tasks however, the system is to find a mapping from stimuli to responses. The latter tasks are more related to the type of learning discussed in the previous section and are the ones that will be dealt with here, since it seems as if neither sensing nor responding alone, will do for a learning system. Also nonassociative tasks can be seen as a special case of the associative tasks where the response is to be invariant to the stimuli. The system's abilities not only to sense but also to respond will be of great relevance to the learning capabilities of the system. This has been illustrated in several experiments in perceptual psychology. The two following examples relate to humans and vision in

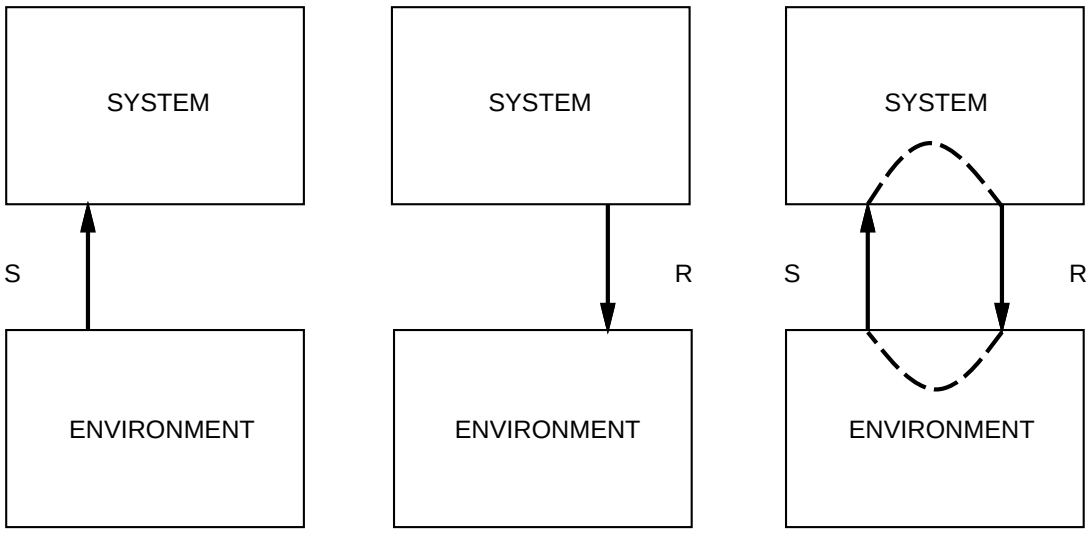

Figure 2.4: Closing the system-environment loop.

particular but should at least point out that responses play a central role in learning visual tasks. They also strongly suggest that closing the *system-environment loop* is necessary for a system to learn successfully, illustrated to the right in fig. 2.4 (Wilson and Knutsson, 1993).

1. In a number of optical rearrangement experiments (Held and Hein, 1958) responses were shown to be necessary for adaptation to take place. Subjects were to view the world through a pair of goggles that optically displaced or rotated the field of view. In one experiment the subject viewed his hand under three different conditions and was then tested for degree of adaptation. First the hand was not moved at all, second the hand was moved by the experimenter and third the subject moved the hand himself. Adaptation to the distortion took place under the last condition but not under the first two.

2. Another goggle experiment was performed were subjects either walked around in the environment for an hour or were moved around sitting in a wheel chair (Held and Bossom, 1961; Mikaelian and Held, 1964). Then adaptation to the distortion due to the goggles were tested. Adaptation again took place when the subject was free to walk but not when only allowed to watch. Compare this with the situations in the left and right of fig. 2.4.

Somehow the system must be told whether or not its responses are advisable. A natural way of doing this is to reward it, whatever this may mean, when it gives good responses, or at least when the task it has been facing is satisfactory completed. Even the behavioristic theories that claim contiguity alone to suffice for generation of associations between stimuli and responses admit that rewards influence outcomes but claim that rewards do not add anything new to associative learning. One of its most vigorous advocators, Guthrie, state the one law of learning as follows (Guthrie, 1935):

> *A combination of stimuli which was accompanied by a movement will on its recurrence tend to be followed by that movement.*

Lax interpreted this means that one learns what one does. With this view it becomes important to make the system do what you want it to do from the very beginning of the learning procedure. Here is an example illustrating the importance of attaching the desired behavior to the proper stimuli or cues (Guthrie, 1935):

> *The mother of a ten year old girl complained to a psychologist that for two years her daughter had annoyed her by a habit of tossing coat and hat on the floor as she entered the house. On a hundred occasions the mother had insisted that the girl pick up her clothing and hang it in its place. These wild ways were changed only after her mother, on advice, began to insist not that the girl pick up the fallen garments from the floor, but that she put them on, return to the street, and re-enter the house, this time removing the coat and hanging it up properly.*

This way of system training is similar to the term *supervised learning* which is the method most frequently used by engineers designing artificial learning systems in terms of *neural networks*. Here the system is trained by being told the correct response to each stimuli. Since this can not be done for all possible stimuli the system will have to generalize from a number of descriptive enough examples chosen by the teacher. When dealing with difficult machine learning tasks it will be impossible to bound the stimuli space with enough examples and the problem will be one of extrapolation rather than interpolation. Standard feed forward neural networks have been shown to yield essentially unpredictable results when required to extrapolate (Geman et al., 1992).

Another class of learning systems are called *unsupervised learning* systems. This relates to the absence of an external teacher to tell the system what to do. Instead the teacher, or learning rule, is fix and built into the system from the start. These systems are often used to learn auto-association i.e. to associate a stimuli with it self, which may be useful in e.g. clustering or pattern recognition tasks. The generality of such a system is rather limited and unsupervised learning can be looked upon as a special form of supervised learning where the teacher is using a predefined learning rule and is built into the system.

The third and last category of learning systems is called *reinforcement learning* systems. Here rewards are used to define tasks for the system. This methodology fits nicely into the behavioristic branch that claim reinforcement to be the main factor involved in association. This branch evolved from a combination of associationistic and hedonistic schools. *Hedonism* refers to the desire for pleasure and the avoidance of pain as to motivate systems to learn. This is also reflected in Thorndike's *law of effect* (Thorndike, 1911):

> *Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they*

> *will be more likely to recur; those which are accompanied or closely followed by discomfort will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.*

The principle bears some resemblance to Darwin's principle of evolution through natural selection, but here it is survival of the most pleasant. The resemblance is however no coincidence since many of the psychologists of the early 20th century were attracted by the work of Darwin. Since the terms pleasure and pain are used within a behavioristic framework they cannot be subjective classifications. Instead what is meant is that states which a subject frequently puts itself in, tries to remain in or does nothing to avoid are called states of pleasure and states which are avoided or not tried to maintain are labeled as painful. This is maybe what Watson means when he at one moment wants to "combat the idea that pleasure and pain has anything to do with habit formation" and then proposes the *frequency principle* stating that the solution to a problem becomes the most frequent response (Watson, 1914). The idea that probable or frequent stimuli-response combinations represents good associations worthy of reinforcement is the very ground of the work here to be described.

2.3 Reinforcement Learning

Instead of defining rewards using the frequency principle as stimuli that the system tries to receive as often as possible, one can work the other way round and define the system from the rewards. When constructing artificial learning systems we can equip them with a single sensory input devoted to receiving payoffs. Now the behavior of the system may be defined by stating, using the law of effect, that rewarded responses become the most frequent ones. This makes it possible to define tasks for the system via the reinforcement stimuli. The duality of the law of effect described above was first discussed by Meehler as an answer to a critique of the law. Critics of the law of effect have argued that it is not an empirical law but rather a definition of a reinforcing event. If a reinforcer is defined as something that strengthens a stimuli-response coupling the law, with this substitution, then states that a stimuli-response pair closely followed by something that strengthens the coupling will receive an increment in its strength. This is nothing but to argue in a circle. Meehl who tackled this problem found that the law is used in two different modes (Meehl, 1950). In an initial "definitial" mode the law is used to discover that a stimuli is a reinforcer, e.g. finding that scratching a dog behind its ears reinforces a desired behavior in a training situation. The other mode is a "predictive" one where the reinforcing stimulus is found to be useful also for reinforcing other behaviors. Reformulated with this in mind the law of effect can be stated as:

> *A reinforcer can be used to increase the probability of* any *learnable response.*

This means that stimuli fall into two different categories, those which can be used as reinforcers and those which can not. To be complete the list should also include stimuli that could be assigned a reinforcing effect by learning, i.e. secondary reinforcers. For a summary of different conjectures on what these stimuli have in common see (Kimble, 1961). As seen the metaphor of reinforcement learning should not be taken to strictly since there is no single sensory input devoted to receiving payoffs in humans or animals.

A major advantage in favor of the reinforcement learning paradigm is its absence of a teacher to tell it exactly what to do in each situation. Consider this example (Valtonen, 1993):

> *Imagine for example the problem of building a machine to play backgammon. One could of course use a supervised learning system and provide it with an expert's move as the desired output. This system can never become better than the expert, maybe more alert, but not really better. A more natural situation is to use the outcome of the game as the reinforcement signal to a learning system. By this we have the possibility to achieve a system better than any expert.*

This approach was used by Tesauro in designing a system that, as in the example above, learns to play backgammon of world champion class (Tesauro, 1990). The possibilities opened by systems that learn and improve their behavior through self play may thrill the imagination. One such example is the strategic computer W.O.P.R in the movie "War games". Here the goal was that the machine should learn how to "win" a nuclear war by simulation and self play. The rules of nuclear war and the "win" criteria are a little bit more subtle in this case than in the game of backgammon though. Since supervised learning is the far most popular training method for artificial learning systems it is useful to make a few comments on how reinforcement learning differ from supervised learning. That learning involves the idea of a system improving its performance over time according to some task dependent measure is compatible with most theoretical views presented earlier. This measure can be thought of as a function defined over the set of possible system behaviors. The function defines a hypersurface on which each point correspond to a particular system behavior. Now to improve its behavior the system has to move towards higher and higher points on the performance surface An important difference between supervised and unsupervised learning lies in what information about its performance the system is fed with. In supervised learning the system has information about the *gradient* of the performance function available. The gradient of a function at a point is a vector pointing in the direction of steepest ascent. Hence the system has *directed* information about how to change its behavior in order to improve its performance, at least locally. The directed information is not often the true gradient itself, but rather an error vector computed as the difference between the desired and actual system responses. The difference to reinforcement learning is that in supervised learning the system is told both *if and how* its performance is to be changed. The problem with this method is, as was mentioned earlier, that the correct answers to the different stimuli have to be known. The learning task has to

be solved from the beginning, at least for some representative cases from which the system can generalize by interpolation.

In reinforcement learning, on the other hand, the teacher does not tell the system anything about the desired response but instead lets it know how good, or bad, it performed. This corresponds to the *value* of the performance function at a given behavioral point, meaning that no directed information is given to the system. This can be understood simply by comparing, in a single response task, the error signal used in a supervised learning system to the credit signal used in a reinforced system. A negative error, i.e. difference between desired and actual output, would tell the system that the output was too high. The critic being negative means nothing in itself. Maybe it is the highest reinforcement that can be given to the system. One way of solving the problem with undirected information is to build a model of the environment that can be differentiated with respect to the reinforcement signal. Differentiating a model establishes the gradient of the reinforcement as a function of the model system parameters. The model can be known a priori and built into the system, or it can be learned and refined during the training of the system. Knowing the gradient of the error means knowing in which direction in the parameter space to search for a better performance. How far to go in the direction of the gradient is however not known, and the step length must be short enough to prevent the system from oscillating. Another way would be to apply optimal control methods, but they are only applicable if there exist an explicit model for the system that is sufficiently accurate, which is not very often the case in a learning situation. Also, when a model is inaccurate, explorative behavior is needed anyway to improve on the model.

From the above it is clear that reinforcement learning is a more general method of learning since any supervised learning task can be reformulated as a reinforcement learning task, e.g. by conversion of an error vector to a monotonous function using the norm of the error vector. The norm could then be used as reinforcement to the system. Note that the reinforcement signal does not have to be scalar. For example, a number of performance criteria has to be taken into consideration when evaluating system behavior in a multicriterion task. In contrast with supervised learning the reinforced system is told *neither if nor how* its performance can be improved. There are many problems where it is difficult or even impossible to tell the system which response is the desired to a given stimuli, or where there may be several correct responses to a single stimuli. It may however be quite easy to decide when the system has succeeded or failed in a task, e.g. to check if a TV set is working properly after being assembled by the system, or to give a statement about some overall measure of the system's performance.

Active exploration is necessary when no model to help estimating the gradient is available. Appropriate actions can only be found by the system performing repeated trials and comparing the outcomes. However, having performed an action the world may have responded and the system now views a different stimuli making it impossible to try new responses to the previous stimuli. This type of learning relates to *trial-and-error* learning studied by psychologists. The term stems from Thorndike who used to call it trial and success and later preferred to call it learning by select-

ing and connecting (Thorndike, 1898). The behavior of an individual is supposed to be reflected in its consequence for receiving reinforcement according to the law of effect. There is a fundamental trade-off between the system's will to explore and exploit. At the hart of this conflict lies the observation that the property of being the better of a number of alternatives is not an intrinsic property of one alternative but a relative one that cannot be found without evaluating and comparing all the alternatives. The cost of exploring, in terms of failures and learning time, must be considered in relation to the learning effect it gives raise to. This trade-off can result in an undesirable tie - a system that neither explores nor exploits. A solution in terms of selective attention, making the system switch between the two behaviors, has been proposed (Thrun and Möller, 1992). To be able to search the performance space, a stochastic behavior component is often introduced. Note that this is necessary also when a model is not known a priori but is to be learned and refined on-line. At first the system may respond in a stochastic way, a behavior that is modified as the system happens to do something that is rewarded. This stochastic behavior can also help the system to avoid getting trapped in local minima. For a class of reinforcement learning tasks in finite state spaces, Whitehead has stated a theorem saying that the time to find an optimal policy when undirected, or stochastic, search techniques are applied is bounded below by an expression exponentially in the size of the state space (Whitehead, 1991). A number of different techniques for directed search has been suggested (Barto and Singh, 1990; Sutton, 1990; Moore, 1990). Two examples are error-based and recency-based exploration. Error-based exploration try to provoke the system into states where it has performed bad in order to improve its knowledge in this domain. In recency-based exploration a dynamical view is taken on the system. Old knowledge is supposed to be inaccurate and hence the system is forced to explore states that has not been visited for a long time. Both of these techniques have a problem in common. To be able to keep track of which states that has and has not been explored this information must be stored somewhere. This will become infeasible when the state space is infinite.

In a complex system that in some way is supposed to improve its performance, there is always a problem in deciding what part of the system that deserves credit or blame for the performance of the whole system. This is called the *credit-assignment problem* (Minsky, 1963) or, to be more specific, the *structural* credit-assignment problem. In supervised learning the desired response is known and this problem can be solved, since the error in each of the response components is known. This is used for instance in the back propagation algorithm for training of feed-forward neural nets (Rumelhart et al., 1986). Another form of the credit assignment problem is referred to as the *temporal* credit assignment problem. This problem arise when the reinforcement is delayed or infrequent. As an example consider the loosing team that scores one goal during the last seconds of a football match. It would not be clever to blame the responses responsible for the last scored point for loosing the game. One solution to this problem is to supply the system with an internal predictor of the gradient of the reinforcement signal and let the system learn to improve its predictions, the so called *adaptive critic method* (Barto et al., 1983). These algorithms were extended by Sutton who also presented some convergence

proofs (Sutton, 1988). The thing in common with these methods is that they all take into account the sequential structure of the input, which classical supervised learning methods do not.

Both methods for coping with the problem of temporal credit assignment bears some resemblances to what psychologists call *secondary reinforcement*. The term was used by e.g. Tollman who claimed that expectancies are the bits of knowledge that are learned (Tollman and Brunswik, 1935). It is called secondary since direct, or *primary reinforcement*, is propagated backwards. A stimuli that through some response leads to a pleasant stimuli will be considered pleasant too. Adaptive critic methods try to estimate the difference between the primary and secondary reinforcement.

The adaptive critic methods not only relates to psychology but also to a mathematical theory known as *dynamic programming*. This theory involves methods for successively approximating optimal performance measures for both deterministic and stochastic control problems (Bertsekas, 1987). A disadvantage with dynamic programming is that the repeated generation and expansion of all possible states is required. In particular there are no convergence theorems for problems with continuous stimuli-response spaces, or for systems involving non-linear evaluation functions as in the neural net system by Tesauro for backgammon (Tesauro, 1990). This example serves to point out that convergence seems to be possible to achieve, at least for a restricted class of problems.

The reinforcement signal must be capable of evaluating the overall performance of the system and be informative enough to allow learning, since this signal is the one and only piece of information available to the system about its performance. The generation and distribution of reinforcements is a subject that in most cases concerning artificial learning systems has not been discussed. In the psychological community these questions have been under investigation and found to have some profound effects on learning and forgetting. The most well known investigator is Skinner who studied *reinforcement schedules* (Ferster and Skinner, 1957). In everyday life reinforcement is neither uniform nor regular, which may be an explanation to why these schedules appear so counter-intuitive. Animals trained to a given behavior with continuous reinforcement (CRF) will, when the distribution of reinforcement is stopped, behave almost as if no training had occurred. Also the response frequency is lower if CRF is given than for any other case. This may be because to reach reinforcement saturation the response frequency has to be higher when reinforcement is received less often. To see this, compare the slope of the plot showing accumulated responses in the case of CRF to the other ones in fig. 2.5. The counter-intuitive appearance of reinforcement scheduling can be summarized as (Walker, 1984):

> *It is more often that is worse, rather than big rewards being worse than small ones, and more often is worse at the end than at the beginning.*

Skinner identified two main categories of reinforcement schedules called *interval* and *ratio* schedules. These can be further divided according to whether they are fix or variable, resulting in the four categories described below. In fig. 2.5 the

cumulative responses before (left) and after reinforcement extinction (right) are plotted. Delivery of reinforcement is indicated with a downward tick. Notice that there are different scales in the two halves of the figure.

Fixed interval (FI) means that the first response after a fixed time interval is rewarded. The interval may be between e.g. onset stimuli, or reinforced responses. Skinner found that the average amount of responses per reinforcement was approximately constant and that the response rate go down to about zero immediately after the reinforcement is delivered.

Variable interval (VI) involves using random interval lengths, which eliminates the response pause present in (FI). Behaviors trained with (VI) are quite resistant to extinction.

Fixed ratio (FR) refers to the number of responses the subject has to perform before being reinforced. Often responses come in bursts because the faster the response the sooner the reinforcement. Again a period of low response rate is found immediately after the delivery of reinforcement. A far fetched analogy would be the student having worked hard to finish his licentiate thesis in time who then finds it hard to start to work again towards his PhD thesis.

Variable ratio (VR) is the schedule used in Las Vegas, with low probabilities for a single response to be rewarded. Here the number of responses needed for reinforcement is varied randomly, giving an approximately constant probability for reinforcement. The result is a high and uniform response rate since this will lead to reinforcement more quickly. This schedule is the one most resistant to extinction of reinforcement.

As seen, the generation and delivery of reinforcement can be crucial to the system's learning capabilities. The reinforcement should give some hint of whether or not the system is on the right track in solving the problem, even if the best solution is far away. One way is to make the reinforcement signal adaptive, so that it initially gives a great deal of credit for a relative moderate improvement, but gets harder in its critic as the system improves its behavior. This method of modifying behavior by successive approximation is called *shaping* in the psychological community. In some cases it is obvious how to choose the reinforcement signal. Never the less, many times it is not clear how to measure the performance, and the choice of reinforcement signal will affect the learning capabilities of the system. How is the current performance of the system to be evaluated if the goal concerns properties of future actions? One approach is to use an adaptive subgoal performance measure (Mendel and McLaren, 1970). The idea is that maximizing the subgoal performance measure will lead to the goal. The subgoal measure is adaptive which means that the system itself should learn a measure that is consistent with the goal at hand. Similar ideas have been used by psychologists such as Skinner under the name of *response chains*. Skinner argued that most complex behaviors can be synthesized from simple stimuli-response pairs. The common way to achieve this is to work backwards starting with the behavior leading to the ultimate goal and then prepending the

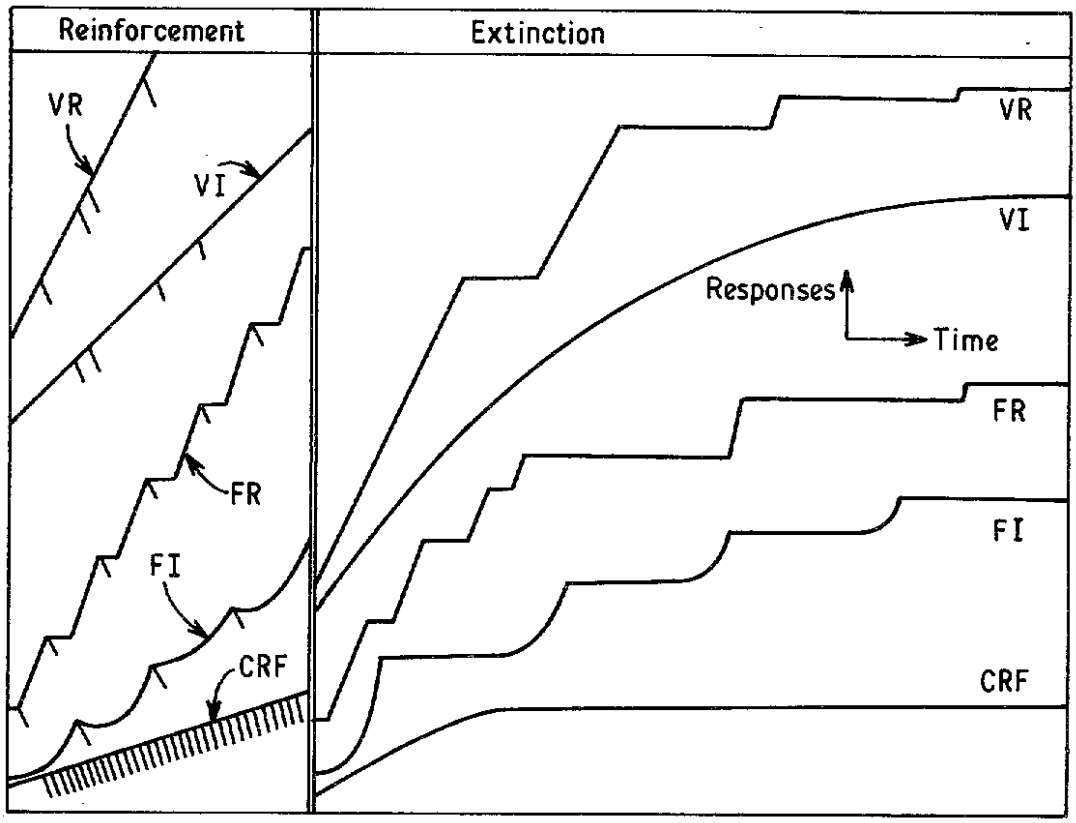

Figure 2.5: Reinforcement schedules. Illustration from (Walker, 1984)

chain with new subgoals to be reached before reinforcement is delivered. A typical example is illustrated in an experiment with one of the numerous rats trained by Skinner (Skinner, 1938):

> *The behavior consists of pulling a string to obtain a marble from a rack, picking up the marble with the forepaws, carrying it to a tube projecting two inches above the floor of the cage, and dropping it inside. Every step in the process had to be worked out by a series of approximations, since the component responses were not in the original repertoire of the rat.*

Yet another issue is whether or not also to use negative reinforcement when training the system. Thorndike repudiated a law of weakening by annoying aftereffects. Punishment as he saw it does effect learning but only indirectly, e.g. by leading the subject to do something different from the punished response in presence of the annoying stimuli (Thorndike, 1932). Skinner did not transfer his view of punishment, as the opposite of reward, to the consequence of punishment. He considered punishment as ineffective, since it is only temporary without lasting effects (Skinner, 1938). This is not a too convincing argument since the same thing could also be said about reinforcement because of the response extinction that appear when reinforcement is withdrawn. Studies by Azrin and Holz has shown that punishment under some constraints gives rise to the opposite effects of reinforcement (Azrin and Holz, 1966).

Nonetheless, it seems as if there is more information in a positive reinforcement than in a negative one. Consider teaching a behavior from positive versus negative examples. In most cases the desired behavior spans over fewer stimuli-response pairs than do the undesired ones. Clearly being told not to give a certain response does not contain any information about what do to instead, at least in tasks involving more than two possible responses to a stimuli. This implies that more information is needed to get a desired behavior from telling the system what not to do than telling it what to do. Also negative reinforcement may lead to a passive non-explorative behavior where the system does not dare to find out what to do in order not to be punished.

2.4 Conclusions

From the discussion in this section it can be concluded that the basic idea of learning by reinforcement, based on the theory of behaviorism, is the most general approach to design learning systems which can improve their behavior beyond that of their teacher. Defining the reinforcement function will be the way of describing the problem to be solved to the machine. Little attention has been paid to the question of how to combine the distribution of reinforcement with the design of the learning system, in order to make training as effective as possible. The proposed behavioristic view is one where stimuli and responses, treated together as decisions, are seen as generated from an optimal behavior distribution, the OBD, when the system is fully trained. This distribution is the solution to the problem the machine is about to learn. In the next chapter a number of different data structures possible for representation of the OBD will be investigated.

Chapter 3

Representation of Behavior

This chapter will deal with a number of possible ways to represent the behavior of a learning system. The structures should be possible to use for reinforcement learning. Many of them are used most frequently in supervised learning systems but will be seen to be useful also for reinforcement learning. In this discussion the world is considered to be fully observable and no information is lost due to limiting system sensors. This view could be widened e.g. by modeling the stimuli as a probabilistic projection of the world state with noise as the probabilistic factor. Limited sensor information can result in several world states appearing identical to the system. Studies of this problem, referred to as *perceptual aliasing*, has been presented but will not considered in this text (Mozer and Bachrach, 1990; Whitehead and Ballard, 1990). Not only should the probability function, $p(\mathbf{x})$, for the stimuli-response pairs, called decisions $\mathbf{x} = (\mathbf{s}, \mathbf{r})$, be represented but also, at least indirectly, the conditioned distribution $p(\mathbf{r}|\mathbf{s})$. The conditioned distribution will be proportional to the distribution of decisions where the stimuli is fix.

$$p(\mathbf{r}|\mathbf{s}) = \frac{p(\mathbf{s}, \mathbf{r})}{p(\mathbf{s})} \propto p(\mathbf{s}, \mathbf{r}) \tag{3.1}$$

Hence representing $p(\mathbf{s}, \mathbf{r})$, will allow for a response to be generated to the current stimuli $\mathbf{s}_0$, by selecting a response at random from the distribution $p(\mathbf{s}_0, \mathbf{r})$.

The response generation could be done using at least two different approaches. One is to let the system model the average mapping from stimuli to responses and then apply some distribution with this function as its mean value. The parameters of the distribution may vary with the stimuli so that the system can be more or less certain on whether the mean value is appropriate or not. Another approach is to let the system estimate the distribution of the decisions on an analytic form so that the conditioned distribution can be calculated once the stimuli is known. An advantage of this approach is that the conditioned distribution may be multimodal, which makes it possible for the system to handle cases when two or more responses are considered equally good.

A choice of approach will inevitable introduce more or less bias. In chapter 2 it was concluded that as little bias as possible should be introduced into a general learning system. However, if the system is to do a specific task it should of course

be allowed to bias the system with knowledge and structures that are known to be of great value for speeding up the learning procedure. The discoveries made by evolution were not made in a day. Care should be taken when introducing biological features as system bias, not to end up with airplanes flapping their wings. The following two properties of the world seem harmless enough to be used to bias the system models and its architecture (Granlund and Knutsson, 1983; Omohundro, 1992).

- Continuity
- Locality

The continuity prior is that the world is geometric and continuous models are to be preferred before discontinuous ones, as long as the data does not contradict it. The second prior states that models are preferred which decomposes experiences into components which are directly affected only by a small number of model parameters. Only a small portion of the experience base will be relevant to any specific situation, i.e. models should be localized in time and space.

No matter which approach is used the system will have to estimate model parameters. Parameter fitting usually results in good generalization but has a fundamental problem in overfitting, i.e. having insufficient data to validate the model at hand as useful for generalization. A solution to the problem of overfitting is to start with a small number of coarse models, when only a small amount of data is available. Then, as more data arrives, more complex models could be validated. Note that if some performance criteria is available it is possible to stop the model complexity from growing larger than necessary. This makes it possible to vary the model complexity across the signal space. A learning system should be able to smoothly move from a memory based regime where the data is the model towards ever more complex parameterized models. Because of the locality prior, model components only affect a subset of the data. The complexity of the model can hence be chosen according to the data that has been received in a specific region, allowing for high model complexity in some part of the signal space and low complexity in another one. If more data arrives and regularities are discovered, usage of more complex models can be justified.

In the following presentation it will be shown how each of the two different approaches, learning an average mapping of stimuli onto responses or the distribution of decisions, can be implemented using two system architecture principles. A popular way of having a system learn an arbitrary function or probability density function is to make use of neural nets. Neural nets can be used in both the above approaches. Another methodology which is also applicable to both approaches is to use various forms of look-up structures. This means that the signal space, embedding the stimuli in the first approach and the decisions in the second one, is decomposed into subsets. These subsets then contain a model of the signal in this subset of the signal space. The stimuli vector is used as a pointer to a memory location that contains the response to that input, or contains a probability distribution from which the response is selected at random.

3.1 Neural Networks

The basic component of any neural network is the neuron. One important neural networks structure is the multilayer perceptron *MLP* , a neural network which has variants of a neuron called *perceptron*, as its basic element (Rosenblatt, 1958). In the original formulation the scalar product between the input signal $\mathbf{s}$, and a weight vector $\mathbf{w}$, was fed through a hard limiting function f, to produce the output r.

$$\begin{aligned} u &= \mathbf{w}^T\mathbf{s} \\ r &= f(u) \end{aligned} \tag{3.2}$$

One single neuron is only capable of dividing the input space into two halves and vary according to the nonlinearity across the border. When putting several neurons together in a network more complex mappings can be implemented. In the first approach the distribution of responses to a given stimuli is modeled as

$$p(\mathbf{r}|\mathbf{s}) = g(\mathbf{m}(\mathbf{s}), p_1(\mathbf{s}), \dots, p_m(\mathbf{s})) \tag{3.3}$$

A network is then to implement the function giving the mean response vector $\mathbf{m}$ of some distribution g. The parameters p_i of the distribution are often some function of the performance of the net for a given stimuli, e.g. depending on the received reinforcement in similar situations. As an example consider a system with a one-dimensional response. The distribution g may be chosen normal, with mean value $m(\mathbf{s})$ and the single parameter p_1 corresponding to the standard deviation. This results in a sort of running gauss with different standard deviations at different points in the stimuli space, depending on how certain the system is on its behavior. A network consists of one or more layers of neurons. All but the last layer in the net are usually referred to as *hidden layers*. Often all neurons in one layer take inputs from all neurons in the previous layer resulting in a fully connected network. In the last layer the nonlinear function is often replaced with a linear function which tend to make learning easier (Hush and Horne, 1993). The function of the network is described in fig. 3.1 and in eq. 3.5 below. In this equation a function $\mathbf{f}_k = (f_{k1}, \dots f_{kn})$ is a vector functions in which each component f_{ki} corresponds to the function of a single neuron in the net, see eq. 3.3. It can e.g. be a hardlimiting function as was the case in the original formulation of the perceptron. This means that the dimension of the vector $\mathbf{u}$ is equal to the number of neurons in a hidden layer. The application of the linear transformation $\mathbf{W}$ to the vector $\mathbf{u}$ produces the net output $\mathbf{m}$.

$$\begin{aligned} \mathbf{u}(\mathbf{s}) &= \mathbf{f}_n(\mathbf{W}_n\mathbf{f}_{n-1}(\mathbf{W}_{n-1}\cdots\mathbf{f}_1(\mathbf{W}_1\mathbf{s})\cdots)) \\ \mathbf{m}(\mathbf{s}) &= \mathbf{W}_{n+1}\mathbf{u}(\mathbf{s}) \end{aligned} \tag{3.4}$$

The basis functions f_{ki}, in the MLP have global *receptive fields*. The receptive field is the region of the input signal space from which the neuron gets its information, see fig. 3.2. A common choice is the *sigmoid function* in eq. 3.5. In two and more

dimensions the nonlinearity will be along a direction in space, as is illustrated in fig. 3.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

It has been shown that any nonlinear mapping can be approximated arbitrary close using an MLP with two hidden layers (Cybenko, 1989). This may however call for the use of infinitely many neurons in the two layers. There are problems requiring an exponential number of neurons implemented using a 2-layer net which has been shown to be possible to reduce to be polynomial when using a 3-layer net (Hajnal et al., 1987). If this is also true for networks with more than three layers is an open question (Hush and Horne, 1993).

Even if it the MLP is capable of implementing any nonlinear mapping it is the possibility of having the net learn this mapping that makes it interesting. The lack of working learning algorithms for the MLP made the research area fall asleep in the mid sixties. A solution in terms of gradient search algorithms was found after about a decade and again neural nets become an attractive research area. To make learning rules based on a gradient search operate it was necessary to exchange the hardlimiting function for a differentiable, continuous, monotonically increasing function that is bounded below and above. One such function is the sigmoid function described earlier. The most popular gradient based algorithm for supervised learning systems is the backpropagation algorithm where errors in the output layer components are used to adjust the weights in the hidden layers in a recursive manner (Rumelhart et al., 1986). For reinforcement learning systems there are learning algorithms similar to backpropagation that, on average, perform gradient search in the weight space of the net (Williams, 1987). In tasks involving temporal credit assignment problems the TD method discussed earlier could be used. Convergence in the general case is however not guaranteed when neural nets are used to represent the reinforcement predictor (Bradtke, 1993). As seen, an MLP can be used to implement any nonlinear mapping. The problem of finding the optimal set of weights which performs the mapping exactly turns out to be NP complete (Judd, 1990). If settled with a local optimum a gradient based search algorithm such as the backpropagation algorithm can be used. Gradient descent algorithms are however very slow when applied to MLPs. This is because the error surface for MLPs often tend to have large amount of flatness with steeps in between. A large step length would be preferable in the

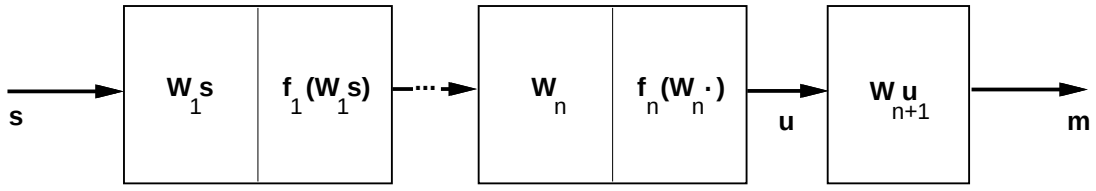

Figure 3.1: An MLP network with n hidden layers each consisting of one linear and one non-linear part.

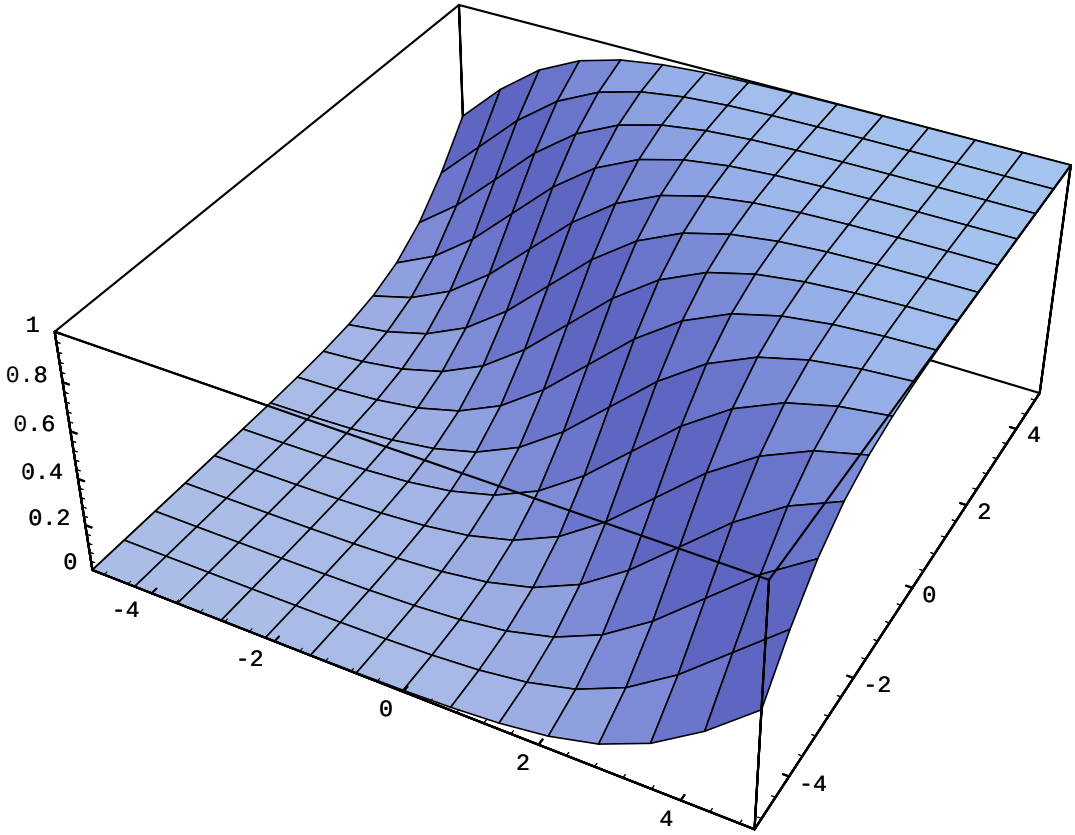

Figure 3.2: A sigmoid basis function with a global receptive field.

flat areas but would make the algorithm unstable when it reached the steep parts of the surface.

Before applying any learning rule a number of other issues must be investigated. One is how large the net must be to approximate the mapping as closely as wanted. If the net is too small it will not have the capabilities to model the mapping close enough. On the other hand, if the size is to large the net may be capable of implementing a number of solutions. All of them may be consistent with the experienced data, but result in poor approximation in between these examples. A number of network growth and prune techniques have been presented as solutions to this problem (Friedman and Stuetzle, 1981; le Cun et al., 1990). Some rules of thumb says that no more than three layers should be needed, and that two layers suffice in most cases. For a 2-layer net it has been shown that to memorize, exactly, an experience set of size n the same order $O(n)$ of neurons is needed (Omohundro, 1987). This indicates that no more neurons than there are experienced decisions should be used in the hidden layers, since that would result in poor generalization. The net would then simply memorize the experienced data.

The sluggishness of the learning procedure will be worse when the number of weights grow in the net. The number of weights does not only affect the learning rate,

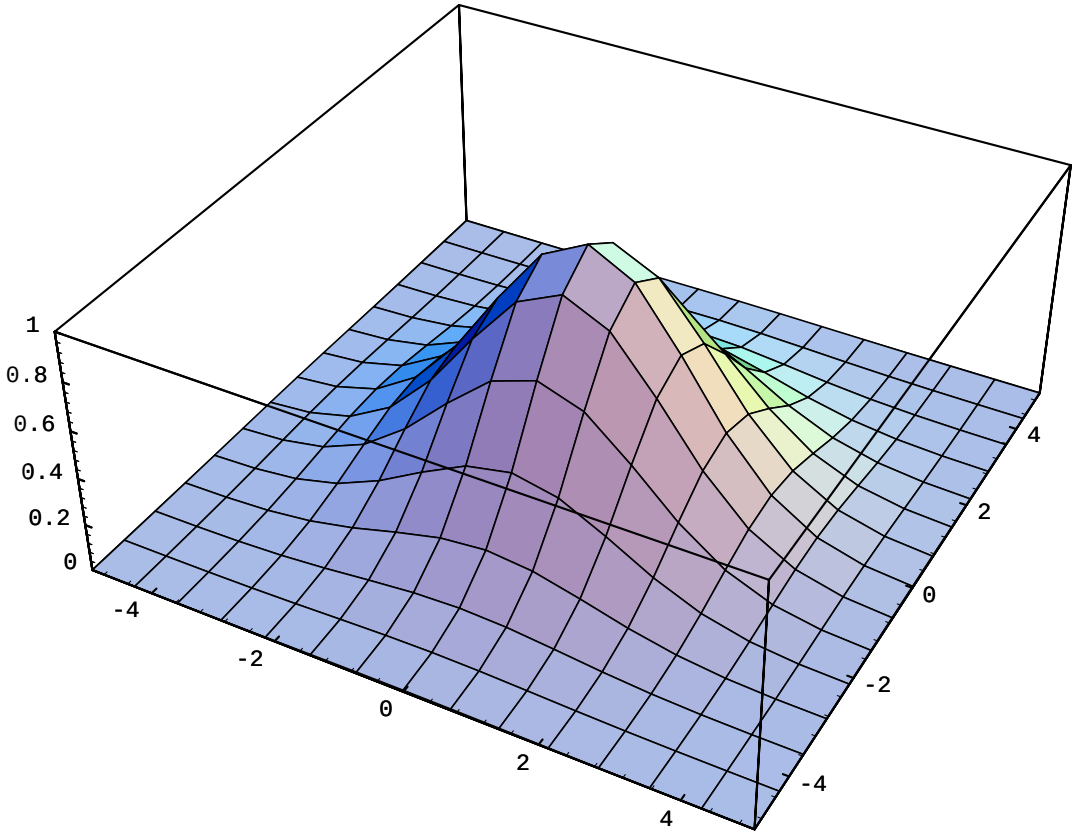

Figure 3.3: A gaussian basis function with a localized receptive field.

but also the generalization capabilities of the network. This issue will be discussed further in section 3.3 where the number of neurons needed for good generalization in a worst case situation is presented. A number of methods have been proposed to reduce the number of weights, without imposing negative effects on the performance of the net (le Cun et al., 1990).

Another type of neural net suitable for learning the mapping of stimuli onto responses arise if the basis functions of the neurons are changed. Nets with *radially symmetric basis functions* are often referred to as RBFs. These networks only have one single hidden layer containing basis functions with localized receptive fields, see fig. 3.3. The outputs from this single layer of basis functions are then put together using a linear combination. This can be seen as the two step procedure where first the input vector $\mathbf{s}$ is fed through the vector function $\mathbf{f}(\mathbf{s}) = (f_1, \ldots f_m)$ producing a new vector $\mathbf{u}$. The matrix $\mathbf{W}$, containing one weight vector per output component, is then applied to the vector $\mathbf{u}$ and forms the output vector $\mathbf{m}$. As was the case with the MLP, the vector function $\mathbf{f}$ again produces one component for each basis function. This time the dimension of the vector $\mathbf{u}$ equals the number of radial

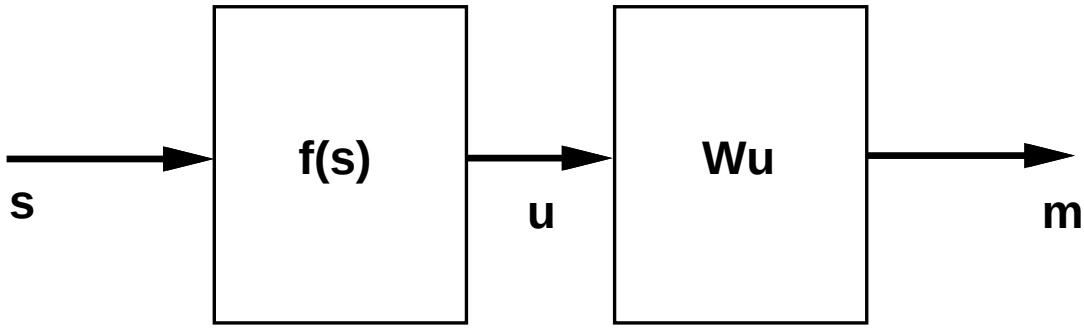

Figure 3.4: An RBF network.

functions f_i, that are used in the approximation of the mapping.

$$\begin{aligned} \mathbf{u}(\mathbf{s}) &= \mathbf{f}(\mathbf{s}) \\ \mathbf{m}(\mathbf{s}) &= \mathbf{W}\mathbf{u}(\mathbf{s}) \end{aligned} \tag{3.6}$$

These types of networks are also capable of approximating any continuous nonlinear function. The RBF implementation may be thought of as simpler than the MLP since it only uses one hidden layer. This is a bit misleading since some problems fit very nicely into the MLP implementation, while some are more easily solved with an RBF net. Consider e.g. the task where two linearly separable classes are to be distinguished. This could be done using a single neuron with a global receptive field but may call for several neurons with localized basis functions to cover one of the class regions dense enough.

The learning phase for RBFs is usually decomposed into two steps. First the hidden layer parameters, e.g. mean vectors and covariance matrices, are updated using some sort of clustering technique. One such choice would be to make use of a self-organizing map (Kohonen, 1989). The positions of the basis functions are then distributed according to the statistics of the decision vector. More basis functions are positioned in areas where decisions are probable, than in low probability areas. Then the parameters for the linear combination are computed, e.g. by mean square error minimization techniques. A number of techniques for adding or deleting nodes to and from the net has been proposed, as was the case for MLP nets. The top down way of doing this is to start with a few nodes and then add nodes when the net is considered to need them. The other way is to prune or remove nodes from a net that starts off with a maximum number of nodes in a bottom up fashion. Learning times for RBF nets have been shown to be shorter than for MLPs (Moody and Darken, 1989). This is due to the two step principle used in the training of the RBF networks. The clustering problem which is faced in the first phase is however NP complete, as was the problem of finding optimal weights for the MLP net. Again some gradient based algorithms may be used to make the learning converge in a reasonable amount of time, but with the possibility to end up in a local optimum. The second stage is on the other hand only polynomial, since it involves finding a matrix inverse as its basic component.

The second way to use the neural net approach is to estimate the decision distribution $p(\mathbf{s}, \mathbf{r})$ directly. The RBFs seems as the most natural net model to use for this task since it fits nicely into the *kernel methods* for density estimation developed in the statistical community. The density function is estimated by summation of a number of kernel functions which can be thought of us bumps. Usually the kernels are placed at the sampled observations of the distribution. When used in the framework discussed here the observations correspond to experienced decisions. Instead of placing kernels at each observation, a smaller number of kernels may be used in the estimate of the density function. In this case the same cluster algorithms which were used for learning RBF networks may be used to determine how many kernels that are needed, where to place them, and their parameter values. Then the weights to use in the linear combination of the kernels can be found by a least square error fit to the observations. That would correspond to the second phase in the learning procedure for RBF nets.

Define a kernel as a function with two arguments and satisfying the following conditions:

$$\int f(\mathbf{x}, \mathbf{y})\, d\mathbf{y} = 1 \tag{3.7}$$

$$f(\mathbf{x}, \mathbf{y}) \geq 0 \quad \forall\, \mathbf{x}, \mathbf{y} \tag{3.8}$$

Now $f(\mathbf{x}, \cdot)$ can be thought of as a kernel function positioned at $\mathbf{x}$. The constraints presented in eq. 3.7 and eq. 3.8 will ensure that the estimated density function, produced as a linear combination of the kernel functions, will also be a distribution. This is a somewhat extended formulation, since in most presentations the kernels are used summed up not weighted together.

$$p(\mathbf{s}, \mathbf{r}) = \frac{1}{\sum w_i} \sum_{i=1}^{n} w_i \cdot f(\mathbf{x}_i, (\mathbf{s}, \mathbf{r})) \tag{3.9}$$

Smoothness properties of the estimate will be inherited from those of the kernel functions $w(\mathbf{x}, \cdot)$. An example of a distribution estimated with four kernels is found in fig. 3.5.

The kernels need not be radially symmetric, and this is not necessary for the algorithms used to train an RBF to work either. Another criteria is that the kernel function $f(\cdot, (\mathbf{s}, \mathbf{r})$ must be possible to use for random generation of responses. As an example consider estimating the probability density function of a system with a one-dimensional response using normals $N(\mathbf{m}, \sigma_s, \sigma_r)$, as kernel functions. That would result in a distribution on the following form:

$$p(\mathbf{s}, r) = \frac{1}{n} \sum_{i=1}^{n} p(\mathbf{x}_i, \sigma_{si}, \sigma_{ri}) \tag{3.10}$$

Fixating the stimuli in the multi-dimensional normal distribution $p(\mathbf{s}, r)$ will result in a new sum of normal distributions $N(\mu_i, \sigma_{ri}^{'})$, since a normal distribution must be normal in all its components. The corresponding response can hence be found by

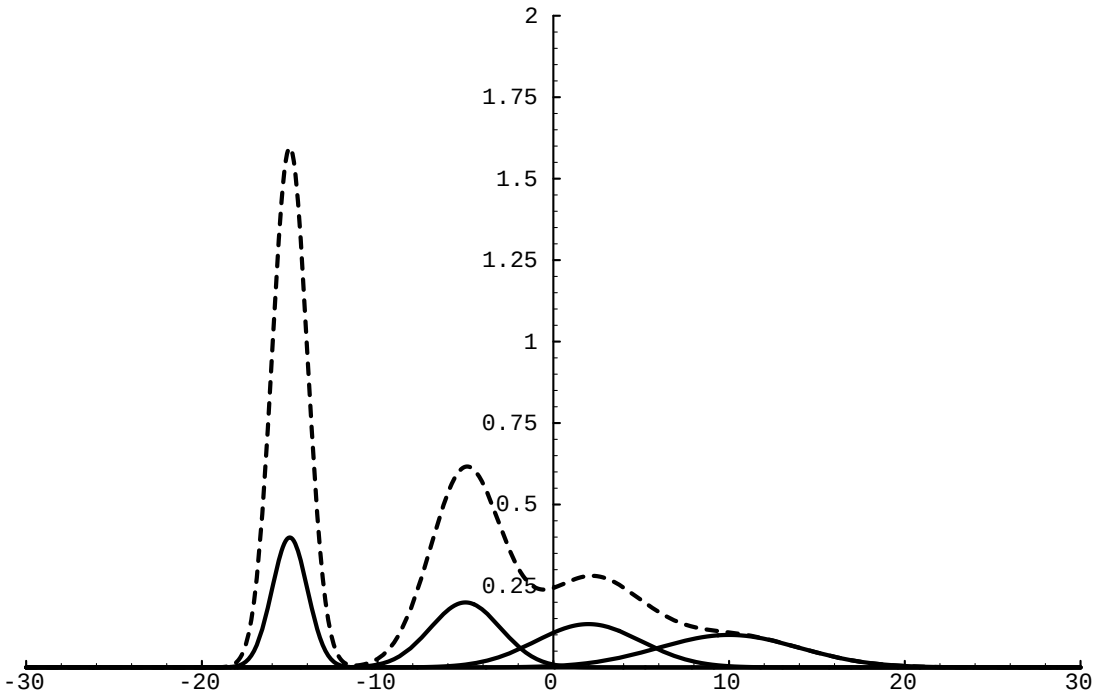

Figure 3.5: Function resulting from a weighted summation of four kernel functions. To have the function represent a valid probability distribution it need to be normalized.

random generation from a new sum of normals. This will be illustrated more precise in the next chapter where normal distributions are used as kernels in a density estimation approach.

3.2 Look-up Structures

A lot of effort has been spent on finding data structures with good worst case performance in the limit when the size of a representational problem grows arbitrary large. Intricate and complex structures are needed to deal with these worst case situations. Such situations are often extremely rare in practice and the structures tend to be hard to implement and inefficient for realistic problem sizes. Instead this section will treat data structures that are simple and show good performance on average. The structures will be used to store information about the distribution of stimuli-response pairs in a way that will be easy to access by using various parts of the decision vector. The components of this vector will act as a key, or address, to the data structure. This could be seen as a memory structure much like the one found in digital computers of today. The decision space must be quantized since the memory addresses are discrete and there is a finite number of memory locations. Different data structures will decompose this address, or decision, space

into subsets in different ways resulting in different organizations of the memory. The quantization can be seen as a matter of system design, and be fix for a given problem or learned by a system having a fixed number of memory locations to play with (Granlund and Knutsson, 1990). The goal here will be to build structures that adapt themselves to the underlying distribution so as to support fast access with small structures.

The first approach partitions the stimuli space and learns the mean response function $\mathbf{m}(\mathbf{s})$ to be used as a parameter to the distribution g, from which the responses are generated. The function for mean response calculation can be implemented by storing models of the response in the data structure. When the stimuli vector is used as an address to the structure it will first point out the region of the stimuli space within which it belongs. At this address the corresponding response model valid in that region of the stimuli space will be stored. The models may range from simple constants up to more complex functions with a number of parameters, as was the case with the RBF. The estimated distribution is produced by adding up the model functions from all over the stimuli space.

$$p(\mathbf{r}|\mathbf{s}) = \sum_{i=1}^{n} g(\mathbf{m}_i(\mathbf{s}), p_{i1}(\mathbf{s}), \ldots, p_{im}(\mathbf{s})) \tag{3.11}$$

This estimation procedure bears resemblances to the use of regression trees such as e.g. CART and AID (Morgan and Sonquist, 1963; Friedman, 1979). These are built by partitioning the stimuli space into leaves with a sequence of binary splits. The goal of the regression procedure is to build a response predictor $d(\mathbf{s})$ from the experienced data. Once the predictor for the mean response vector is found, the parameters p_i can again be used to control the certainty of the prediction by acting as deviation parameters to e.g. a normal distribution. The predictor in CART is constant over each leaf region in the stimuli space. The leaf constant is calculated as the mean of the responses corresponding to the stimuli that fall within the leaf region. This will minimize the error within each leaf in a mean square sense. If supervised learning is used a node is split so as to maximize the decrease in the prediction error. Whether to split or not if reinforcement learning is applied will instead have to be based on changes in the rewards received by the system due to the split. No robust and general stop criteria has been found, at least for the supervised situation, even though a lot of effort has been spent on the issue (Breiman et al., 1984). Consider a regression tree using mean square error as its performance measure. Such a tree will always decrease its prediction error by splitting nodes. The problem remains even if performance is measured in terms of rewards. Even though a split may seem worthless according to the performance measure, it may be that splitting one more time will increase system performance. The main stream solution to this problem is to first grow a tree that is much too big, and then prune away branches as to minimize a cost-complexity measure. Such a methodology is an example of the use of Occam's razor (Blumer et al., 1987; Buntine, 1990) which is a complexity criteria stating that the simplest model consistent with the data should be used. Note that a poorly chosen Occam criteria can result in an algorithm that:

> *like the drunk who, having lost his keys further down the road, only searches for them around the lamp post because that is the place where the light is strongest.*

The way to partition the space and the tree structure presented in the example above is only one of many. Later in this section a number of different structures for partitioning the stimuli space will be reviewed.

Instead of estimating the mean value function for each partition of the stimuli space, the distribution $p(\mathbf{s}, \mathbf{r})$ itself may be estimated in each part of the decision space. This is the second approach using look-up structures. An example of this approach is the bump tree structure (Omohundro, 1991) where each leaf in a tree corresponds to a kernel function. There are also functions associated with each internal node with the constraint that a function in an interior node must be everywhere larger than the functions of its children. Typical examples of functions that lend themselves to easy incorporation in the bump tree structure are the gaussian and the quadric functions. Branch and bound techniques can then be used to answer questions such as a request for all leaf functions with functions values, at a specific point lying within a specified factor of the leaf function with the largest function value. The search proceeds down the most promising branch first to find a leaf function value. Now all subtrees with function values less than the specified factor times the found leaf function value can be pruned away. This procedure is repeated, maintaining the largest node function value for comparison, until no more subtrees are left to investigate. The effect of branch and bound techniques on system performance will be discussed in section 3.4.

This strategy may be used together with any of the structures presented later in this section and corresponds to density estimation using adaptive kernels, as described earlier in this section. Node and leaf functions, $f_i(\cdot)$ can be combined in a convex way either by linear summation or by the use of adpative weights, or influence functions w_i, that are associated with each model function and localizes its influence to certain parts of the signal space. The model function parameters can be determined e.g. by a least square fit of the data close to the center of their corresponding influence function. This fit is achieved by weighting experienced data with the value of the influence function so that data close to the model center affect the solution more than do data farther away. Compare this fit procedure to that required for the RBF where a more expensive global fit of the data is performed. Again decisions are denoted by $\mathbf{x} = (\mathbf{s}, \mathbf{r})$ in the expression for the estimated distribution:

$$p(\mathbf{x}) = \frac{1}{\sum w_i(\mathbf{x})} \sum_{i=1}^{n} w_i(\mathbf{x}) f_i(\mathbf{x}) \tag{3.12}$$

Both approaches presented above can make use of any structure partitioning a signal space. The fundamental idea with all structures for space partitioning is that signal space properties should be connected with properties of the represented signal. An incoming stimulus is described in terms of space coordinates which the structure converts to something in terms of the stored response models. Each node represents not only a subset of the signal space but also a set of experienced decisions

in terms of a signal model. This review of a number of such structures follows that of Omohundro (Omohundro, 1987) apart from the description of the k-**e**-d tree which is the novel structure used in the implementation of the learning system in chapter 4.

The simplest subdivision of the decision space is done with a grid, see fig. 3.6 a). Each dimension is uniformly partitioned, but the number of pieces may wary from one dimension to the other. With this structure the space is partitioned into a number of hyperrectangles all with the same shape, resulting in a constant access time. A stimuli or decision vector points out a bucket by its components. Each component will fall into a bucket along a dimension. The indices of these buckets are then used to address an array holding the signal models. Learning algorithms for these type of structures have been investigated e.g. under the name of extended stochastic learning automata (Sutton and Werbos, 1990). When the outcomes are not equally probable throughout the signal space a need for more sophisticated structures arise. In this case having models for parts of the space where decisions never occur is a waste of space and would of course be impossible in a general system. The memory size would become extremely large and make the system suffer from what Bellman called "the curse of dimensionality" (Bellman, 1957).

A more flexible structure is the adaptive grid, found in fig. 3.6 b). Again the models are stored in an multi-dimensional array but here the sizes of the partitions along each axes may vary. This allows for a finer representation of the data in probable areas than in less probable ones. To access a model corresponding to a given signal vector each of its components are used to index the final bucket in each dimension. These indices are then used as the index to the array holding the models. The buckets are hyperrectangular as they were in the grid structure, but here they are allowed to have different shapes. A basic problem with this structure is that a partition along one dimension of a bucket leads to a global effect on the grid, since all buckets intersecting the hyperplane orthogonal to the partition will also be partitioned. This problem gets worse as the dimension of the space increases.

The problems with global partitioning effects is nonexistent if a multi-level grid is used. A fix sized grid is used to partition the space at different levels, see fig 3.6 c). First the whole space is decomposed coarsely by the grid. This partitioning then continues recursively with each partition until the space is covered dense enough with models. Well known structures of this type are the quad-tree and the oct-tree which arise when two- and three-dimensional spaces are decomposed with grids of size 2×2 and $2 \times 2 \times 2$ respectively. In higher dimensions this structure tends to use large amount of memory for a given accuracy of representation.

Modifying the multi-level grid and allowing splits only along one dimension at a time results in a *k-d trie* seen in fig. 3.6 d). Note that the word trie is not a misspelled form of tree, but a data structure of its own (Fredkin, 1960). The k dimensions are still split in half, but each node in the trie now contains information about which dimension d, to split. The key point with this structure is that only dimensions in which the model does not provide good results have to be further partitioned. The fact that the cut location always halve the buckets simplifies the addressing procedure. Whether to proceed left or right at a node is determined by

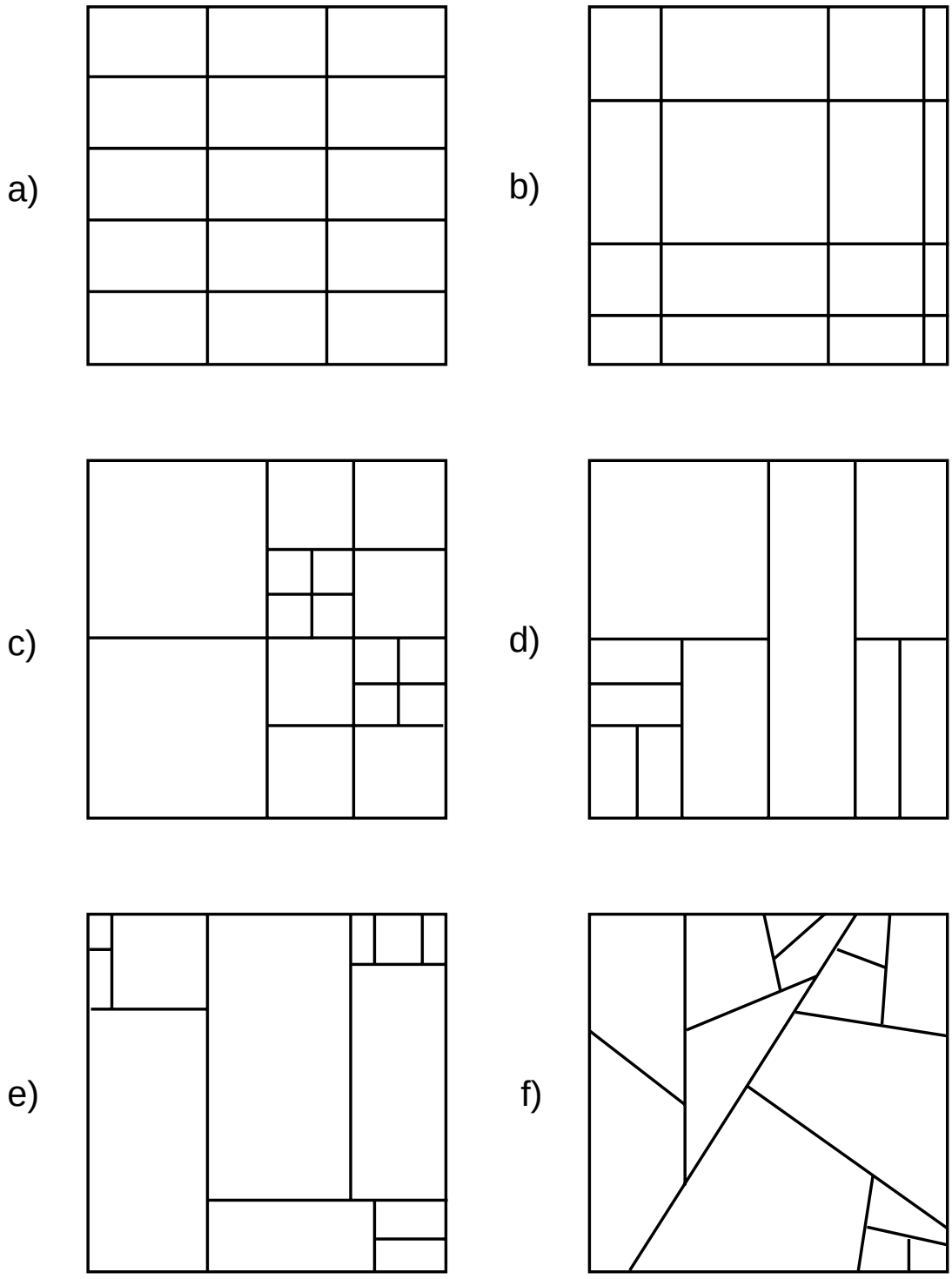

Figure 3.6: Different ways to partition a two dimensional signal space. The partitions are represented by the leaf regions of each structure. a) grid, b) adaptive grid, c) multi-level grid, d) k-d trie, e) k-d tree, f) k-e-d tree.

the bit at position $j+1$ in the signal vector component corresponding to the split dimension. Here j is the number of splits already made along the current dimension.

A natural relaxation of the k-d trie is to allow for splits at any position along a dimension, not just at the midpoint. This is illustrated in fig. 3.6 e). Such a structure is called a k-d tree to distinguish it from trie structures splitting at midpoints. Note that with this notation also quad- and oct-trees should be referred to as tries. The k-d tree is an ordinary binary tree, but each node contains information both about which dimension to split and also where along that dimension to position the split. Each node in a k-d tree corresponds to a hyperrectangular piece of space, in which some or all of the dimensions may be infinite. This holds true even for the leaves of the tree constituting the finest partitioning. Searching for a model in the tree is accomplished by a single traversal of the tree from the root to the leaf. At each node the d:th component of the addressing vector is compared to the stored information about the split location. The traversal then proceeds down the left or the right branch depending on the outcome of the comparison. A balanced tree will have $\log_2 n$ levels, where n is the number of leaf models, which yields a search path of the same length as the depth of the tree, i.e. $\log_2 n$ steps.

The last structure in this review is the k-$\mathbf{e}$-d tree. The partitions in this structure no longer have to be along the dimensions but can be along any direction in space. See fig. 3.6 f). This means that the tree now has to store information both about the direction $\mathbf{e}$ of the split, and also where along this direction the split is made. The vector $\mathbf{e}$ can be seen as the normal vector of the hyperplane that partitions the space at a node. This structure is very flexible and only an extra scalar product between the addressing vector and the normal vector $\mathbf{e}$ has to be done in the retrieval procedure. In the next chapter this structure will be used as the basic component in a learning system.

All the structures in the previous review were built in a top-down manner. Another way of forming them would be to proceed in a bottom-up fashion. One way of implementing the bottom-up strategy is the best-first model merging approach suggested by Omohundro (Omohundro, 1992). The idea is to start with a kernel function at each experienced decision and successively merge model pairs which decrease system performance the least. This means improving a complex model by replacing two of its component models with a single one. This merged model may be of the same complexity as the original components, but since the the combined data from the two original models are used in determining the parameters of the merged model, it is possible to make the merged model more complex than the components from which it was made. The best-first aspect is to always choose to merge the pair of models which decrease the likelihood of the experienced decisions the least. Any common stop criteria could be used, e.g. Occam factors discussed previously, or VCD bounds which will be discussed in the next section.

3.3 Estimating Performance

In the previous section it was suggested that the parameters p_i could be used to control the certainty of an estimation. To do this some kind of system performance measure must be available. Different expressions in terms of received rewards may be used, but there are also other measures available. Before measuring performance it may be of interest to analyze how many correct, or highly reinforced, decisions the system must make to stand a chance of achieving good performance. This is somewhat the opposite to the question of network size discussed before. Then the number of samples said something about the network size, now the size should say something about the number of samples needed. One measure of system performance can be stated as the system's capability to make good generalizations from the data it has experienced. If the system is capable to implement a finite set of functions Q then the generalization $G(q)$, where $q \in Q$, can be defined as e.g. the fraction of all input stimuli for which the system gives correct responses. Two approaches have been proposed for the study of generalization. One is to look at the average generalization of the system and the other is to focus on the worst case of the generalization error.

In the first approach the subset $Q_P \subset Q$ plays a central role. This is the subset of functions which are consistent with the set of highly reinforced decisions P. Highly reinforced decisions are decisions which are accepted as correct. In supervised learning this set is denoted the training set and consists of a number of stimuli together with their correct responses. The average generalization of the system is computed by averaging the generalization measure over all consistent functions $q \in Q_P$. Another approach has however recently gained more interest, since average measures have been found hard to apply in practice.

The second approach tries to establish a bound on the worst case generalization error, i.e. the difference between the generalization on the highly reinforced samples P, and the generalization on the actual task. Worst case measures may not seem as interesting as average ones but may be of severe relevance to certain applications were worst case situations are common, or must not cause system failure. The fact that it seems to be the only measure applicable in practice also makes it interesting.

Since the system has been experiencing the samples P during training, a measure of the error on this set is supposed to be overly optimistic. In many cases it is however possible to bound the difference between this measure and the actual one, and to make it arbitrary small, by adding more data to the training set. A key result, established by Vapnik and Chervonenkis, is that this is possible only when the number of highly reinforced decisions exceeds a parameter called the *Vapnik-Chervonenkis dimension*, the VCD (Vapnik and Chervonenkis, 1971). The VCD will then provide a measure of how capable the system is. The definition of the VCD can be stated as follows:

> *The VCD of a system is the size of the largest set S of data samples which the system is able to divide in all, $2^{|S|}$, possible ways.*

Here $|S|$ denotes the cardinality of the set S. The definition requires only that such

a set of examples S exists, it need not be true for all sets of size VCD. The VCD may be infinite in which case there is not possible to put an upper bound on the generalization error. This is however not to say that it is impossible to achieve good generalization on average. For an MLP with N_l hidden nodes producing binary outputs, the VCD has been shown to be finite:

$$2k\ [\frac{N_l}{2}] \leq \text{VCD} \leq 2N_W \log(e \cdot N_N) \tag{3.13}$$

Here $[\cdot]$ denotes the floor operator returning the largest integer less than its argument, k is the dimension of the stimuli, N_W the total number of weights in the network, e is the base of the natural logarithm, and finally N_N is the total number of nodes in the net. The upper bound on the VCD is valid for any number of layers and regardless of the connectivity in the network. Once the VCD is known the number of samples in the training set can be determined. As a rule of thumb the number of examples should be ten times the VCD (Widrow, 1987). A more exact criteria, valid for classification, is that the number of examples should be of order $O(\frac{N_W}{\epsilon} \log \frac{N_N}{\epsilon})$ to have a system which correctly classifies a fraction $1 - \epsilon$, of the examples in the future. For this to be true it is necessary that the system correctly classified a fraction $1 - \frac{\epsilon}{2}$, of the training examples and that future examples are drawn from the same distribution as the training examples.

The generalization capabilities of an RBF net or any look-up structure using localized kernels in the estimation can, as far as the VCD is used, be described in analogy with the MLP network. The lower bound will be different for different structures while the upper bound is valid for all the structures. The lower bound on the VCD is not equal for the RBF and the MLP. This is because the training of an RBF net is usually a two step procedure where the first step fixates the kernel parameters so that the last procedure cannot fully exploit the complete set of functions that the network is capable of implementing (Hush and Horne, 1993). If the first step is considered as a preprocessing step the bound depending only on the second training procedure can be computed. The last step is identical to training a single perceptron with N_l inputs. A perceptron has a VCD equal to $N_l + 1$, (Baum and Haussler, 1989), which results in the following bounds on the VCD for an RBF network:

$$N_l + 1 \leq \text{VCD} \leq 2N_W \log(e \cdot N_N) \tag{3.14}$$

Here N_N is the number of basis functions and N_W now denotes the number of weights in the linear combination of the basis functions plus the number of parameters in the basis functions themselves. The upper bound on the worst case of the generalization error presented for the MLP did however suppose the output from the hidden layer nodes to be binary. This is not the case in an RBF net, or any look-up structure, with real valued response models. Otherwise the result is valid for any feed forward network. An upper bound for systems producing real valued outputs such as RBF networks, look-up structures, and MLP nets with a linear output layer, is probably a bit higher than that for an MLP with hidden nodes producing binary outputs. A very pessimistic estimate can be computed if the number of bits used for the

continuous output is known. Then the continuous output could be computed using as many systems as there are bits in the output. The result from such a consideration would yield that approximately ten times the number of weights in the system, times the number of bits in the output is the amount of decisions needed for good generalization by the system.

Given that the number of decisions needed for good generalization is experienced by the system, a bound on the worst case error is guaranteed. Another issue is to measure the average system performance, even if it can not be bounded. Next some measures which are often used to estimate performance of systems trained with supervised learning will be studied and their appropriateness for reinforcement learning will be discussed. The most popular error measure is the mean square error. It is defined as the expected error using $p(\mathbf{r}|\mathbf{s})$ to predict responses $\hat{\mathbf{r}}$ and having the set of experienced decisions fix:

$$\text{mse} = \mathcal{E}(\mathbf{r} - \hat{\mathbf{r}})^2 \tag{3.15}$$

How are the experienced decisions to be used both for building a predictor and estimating its performance? The worst estimate of the error is the resubstitution estimate, where all the decisions are used for construction of the system, and then again as ground truth when calculating the performance of the system. Another way is to use a test sample estimate. The training set is then divided into two sets. One is used for construction and one for providing ground truth. The drawback is of course that decisions used as ground truth cannot be used to improve the performance of the system. This problem is dealt with in the cross-validation estimate. Here the training set P is divided into v subsets each containing approximately the same number of decisions, $P = \bigcup_{i=1}^{v} P_i$. Systems are now produced, using each of the subsets P_i. The error estimate for system i is calculated using the data $P \setminus P_i$,that was not involved in the system construction processes, as ground truth. Averaging the v different errors yields the error estimate for the system which is formed by using all the decisions for its construction.

All the measures described above are based on distances and appear to be unsuitable for reinforcement learning, where a decision is accompanied with a reward which itself says something about the system performance at the moment. Such information is not available to supervised learning systems which have to assume something about the metric of the error space. It can happen that the reinforcement the system receives is not proportional to the distance from the response samples which are used as ground truth. The knowledge of how past decisions were rewarded makes it possible to construct more informative performance measures than is the case for supervised learning systems. Instead of distance based measures it is more suitable to use different functions of the received reinforcement $R(t)$, to measure performance. A common performance measure used for reinforcement learning systems is the cumulative reward, R_n.

$$R_n = \sum_{t=0}^{n} f(t)R(t) \tag{3.16}$$

The function $f(t)$ is used to control the influence of rewards at different times, e.g. not to connect rewards received by the system long time ago with its current performance. This estimate could also be accompanied with a certainty measure, e.g. in terms of reinforcement variance, which would say something about the fluctuations in the rewards given to the system.

3.4 Conclusions

In this chapter it has been shown that system behavior in terms of decision probabilities can be represented in two different ways. If as little system bias and as much flexibility as possible is preferred, the approach representing the probability density function itself seems to be the one to choose. To sum up the review of the two system architecture paradigms, neural networks and look-up structures, some differences between them will be discussed. The focus will be on the neural net architecture since it is the one most frequently used. Standard neural networks maintain a complete model of its domain. The model is mostly wrong initially but gets better as more data appear. The net deals with all data in much the same way and has no representation for neither the strength of evidence behind a certain conclusion, nor for missing or uncertain input. The architecture is usually chosen before the data is presented and the processing in the early training phases very much resemble that in later ones. Human and animal learning, on the other hand, seems to proceed in a quite different manner (Omohundro, 1992). When the system has no or few experiences in a domain every single experience is critical. Experiences are remembered more or less in detail in the early phases, and new responses are formed by generalizing from these stored experiences. Later on, when more data is available, more complex models can be formed and there is no longer a need for storing individual experiences. Instead the focus is on discovering regularities in them. The first phase characterized by one-shot learning could be achieved using a look-up structure storing the experienced decisions, while the properties of the second phase resemblance those found in the methods for parameter fitting models.

Many times the choice of the number of hidden layers to use, and how many neurons there should be in them, involves lots of qualified guessing and rules of thumb. Other bounds than the worst case based VCD on how well a given network will perform are not to be found. When using look-up structures, reward based performance measures may be used to determine if and also which parts of the structure to refine. This is possible since such a measure can be made a local one. The training procedure for neural nets also tends to be sensitive to the randomly chosen starting conditions, and has been shown not to scale well with the size of the problem (Tesauro, 1987). As stated before learning rules involving gradient search are generally slow because of the structure of the error surface. Most of the problems can be referred to the use of global receptive fields in the neurons. When the neurons are active in the whole input space most of them will have to be adjusted in the update procedure for every kind of error. There is a tendency to corrupt the performance in one part of the space when correcting an error in another part.

The global receptive fields will not only affect training times. When using neural networks, where the neurons have global receptive fields, all neurons must participate in the computation even though only a few of them contribute to the output. To store n memories in a neural network, the same amount of neurons $O(n)$ is needed (Hopfield, 1982). The computational effort to process an input for a fully connected network capable of exact classification of n memories is hence $O(n^2)$, since the n neurons have to connect to n decision functions in the output layer. To illustrate the overkill, consider a one layer net where each of the neurons defines a hyperplane. The input is processed by letting each neuron determine on which side of the hyperplane the input lies. This results in a waste of computational capacity since many of the comparisons will be unnecessary. The solution is to make use of a structure with local receptive fields so that task complexity may be attacked with the standard method of divide and concur, meaning that every time a piece of information is determined about the data it is used to prune away unnecessary further computations. With hierarchical look-up structures such as bump trees, the computational time can be reduced, at least on average, to $O(\log_2 n)$ using this sort of branch and bound techniques (Friedman et al., 1977). This speeds up the brute force implementation with a factor $n/\log_2 n$ which becomes significant for large n.

But what about the possibility to cut down computational times in neural networks by implementation on parallel hardware? With n processors an optimal speedup of a factor n can be achieved through parallelization. This would result in computational times of order $O(n)$ for parallelized neural nets. If $O(n)$ stimuli are to be processed the look-up structures presented in this section can also benefit from a parallel implementation. In this case computational times, per stimuli, for the look-up structures can be reduced to the order of $O((\log_2 n)^2/n) < O(\log_2 n)$. The factor $(\log_2 n)^2$ stems from the need for a sorting algorithm with times of order $O(\log_2 n)$ to run on each level in a tree of depth $O(\log_2 n)$. Compare this to a parallelized neural net with $O(n)$ connections which needs computational times of order $O(n)$ to process one stimulus. The speed up benefits from parallelization of look-up structures are hence of order $O(n/\log_2 n)$ which is $O(\log_2 n)$ less than the speed up of $O(n)$ gained when parallelizing a neural net.

Even if neural networks benefit more from parallelization than do the look-up algorithms the resulting computational times will still be in favor of the look-up algorithms, since $O((\log_2 n)^2/n) < O(\log_2 n) < O(n)$. In realistic on-line applications the environment will determine what stimuli will arise as an answer to a system response. This will omit the benefits from processing a chunk of $O(n)$ stimuli. Still the unparallelized look-up structures only need computational times of order $O(\log_2 n)$ compared to $O(n)$ for fully parallelized neural nets. Consider a system representing a million memories in line with goal presented in chapter 1. A serial look-up algorithm would take times of order $\log_2 10^6 = 20$ compared to times of order 10^6 needed by a parallelized neural net.

From this it seems as if the more adaptive of the hierarchical look-up structures are to be considered as a promising alternative to standard feed-forward neural nets. In the next chapter a system for reinforcement learning using the k-e-d tree structure is presented.

Chapter 4

A Tree Structure

In complex environments only an exceedingly small fraction of all possible situations will ever occur. However, exhaustive memory space allocation can be avoided by only representing events which are of interest, as described in the previous chapter (Granlund and Knutsson, 1990). This chapter describes how the most flexible of the presented look-up structures, the k-**e**-d tree, can be grown and used for representation of the behavior distribution. Also a reinforcement learning algorithm for a system using this type of structure is presented. To memorize exactly n experiences the storage required is of order $O(kn)$, while the time to grow the tree is of order $O(k^3 n \log_2 n)$. Finally to generate a response using the tree will take times of order $O(k^2 \log_2 n)$.

4.1 Characteristics

The purpose of the tree is to provide an estimate of the system's behavior distribution. This estimate is then conditioned with the current stimuli to provide a distribution of responses, from which the system can generate new responses at random. Experienced decisions provide the raw material from which the tree is grown. The behavior distribution is estimated by partitioning the set of experienced decisions, which are seen as samples from the behavior distribution, and storing local models of the decision distribution in the nodes of a k-**e**-d tree structure. Within each partition the local decision distribution is modeled with a normal distribution, which parameters are stored in the corresponding node in the tree. The idea behind the approach to employ local normal distributions in the approximation originated from the successful use of tensors as local signal descriptors (Knutsson, 1989). An estimate of the total behavior distribution is then formed by summation of the models in the finest partitions, i.e. the models of the leaf distributions. It has been shown that any probability density function, with a finite number of discontinuities, can be approximated arbitrarily close by a sum of normal distributions (Sorenson and Alspach, 1971). As an example consider the case when the optimal behavior is to produce a sine wave. The corresponding OBD is found at the top of fig. 4.1 and an approximation using a sum of four normal distributions is shown at the bottom of the same figure.

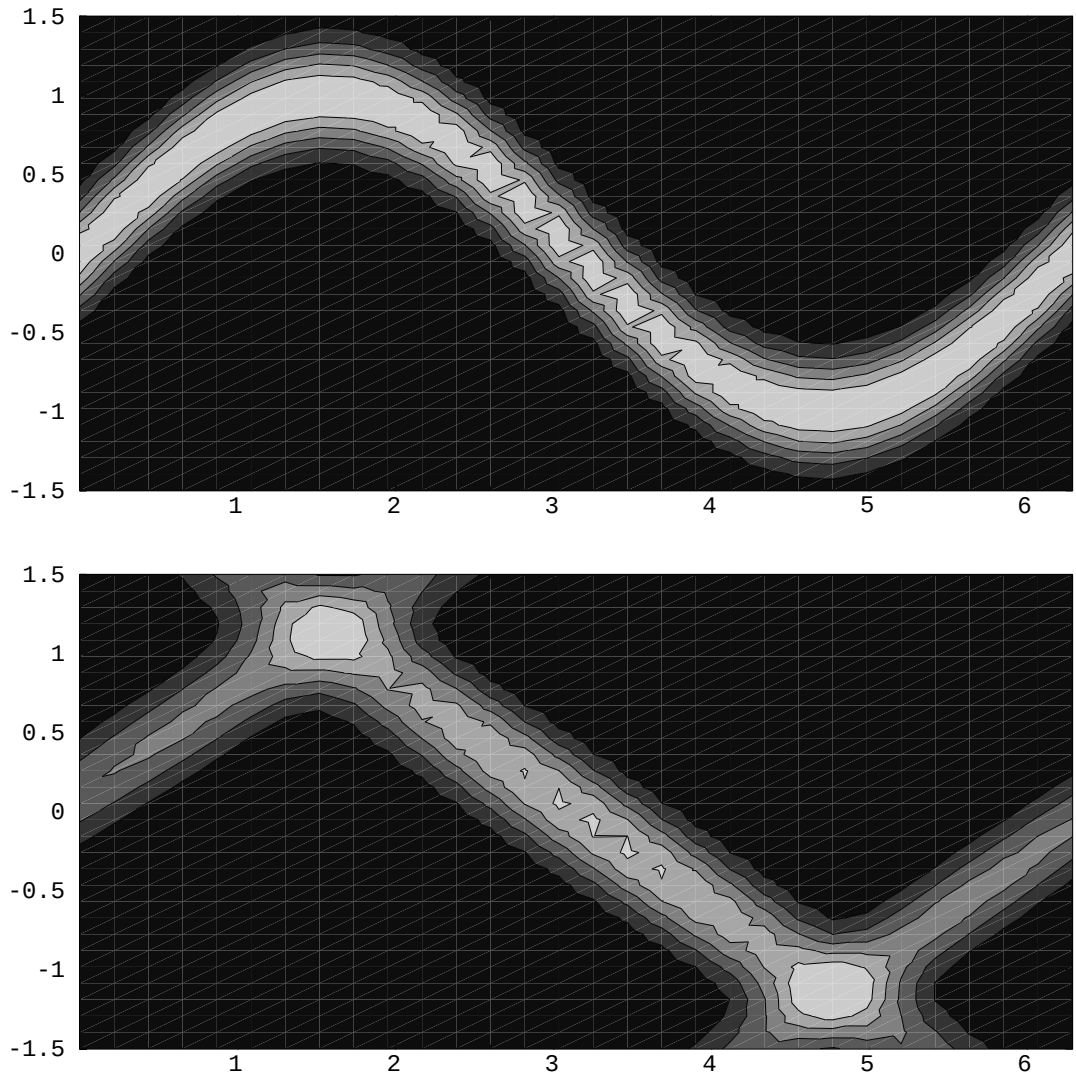

Figure 4.1: Representing an optimal behavior with a probability density function (top). Approximating it using a sum of four normal distributions (bottom).

When the system begins to explore the environment its apprehension of how to behave is rather limited. As more and more decisions are experienced, the amount of raw material for growing the tree will increase. The more data the finer the decision distribution is sampled which allows more complicated models of the distribution to be validated. This in turns creates better opportunities for the system to generate good responses. Since memory capacity is always limited, some criteria for selecting when to memorize an experience is needed. Also a question of which old decision to exchange for a new one arises. Some frequently used rules for computer memory management is to exchange the less recently used or the less frequently used memory. Since models of the decision distribution, and not the decisions themselves, are accessed when retrieving a response, it is not obvious how to incorporate the frequency based rule. A rule similar to the recency based one mentioned above can

however be achieved by marking decisions with the time when they were experienced. Then new decisions, considered valuable enough to memorize, are exchanged for the oldest experiences. This is nothing but the well known first-in-first-out queuing strategy. As a by-product this rule will implement adaptation to changes in the environment, as well as response extinction, since decisions are forgotten if they are not regenerated by the system. Without the ability to forget, problems can arise e.g. when there is noise present in the reinforcement signal, or if the environment varies with time. Bad decisions could be remembered as good ones due to the noise, but since the system will forget these decisions it should be more noise tolerant. Also decisions that once were good but now are bad will face the same destiny. The time label can also be used for more continuous modification of how much a decision is to influence the model calculations.

When is a decision worth memorizing? Since the approach presented here is based on positive examples only, it seems natural to impose some kind of restriction on the reinforcement attached to the decision. A high reward is however not enough reason for storing a decision. If the model, from which the decision was generated, already describes its part of the decision space well enough, there is no need to waste space on making the description any better. Instead, good decisions generated from unsure or bad models are the ones to remember, since they may be used to modify and improve the model. Such a rule is in line with a claim put forward by Tollman, stating that what is learned is what violates the expected. This view was mentioned previously in chapter 2.

Related to these matters is the question of when the number of new responses are considered to be large enough to motivate a new estimation of the behavior distribution. Re-estimation of the decision distribution is the way the system improves upon its behavior. By incorporating new knowledge of how decisions are distributed, the new estimate of the behavior distribution will provide a better ground for the production of highly reinforced decisions. The model of the behavior distribution is represented in the tree structure, which means that a re-estimation of this model will call for a modification of the tree. Now the tree could be regrown e.g. after the production of a good enough decision, or when a certain number or percentage of the stored decisions have been exchanged. Any of these policies should lead to the tree being regrown more often at the beginning of the learning phase, when the system is more or less without any knowledge of what to do, than later on when the system has more experiences to base its response generations on. The rest of this section will discuss issues relevant to the procedure of growing a tree from a set of experienced decisions.

Because the distribution models are localized in the decision space, they will rarely be of equal importance when a new response is to be generated. Such considerations is made by looking at the coefficients in front of the leaf distributions, which sum up to the conditioned response distribution. If a coefficient is sufficiently small, the corresponding model can be considered not to influence the response distribution. This means that branches of the tree, which corresponds to models considered not to influence the generation of a response, can be cut away from further considerations. The result is lots of saved work. One of the major advantages with

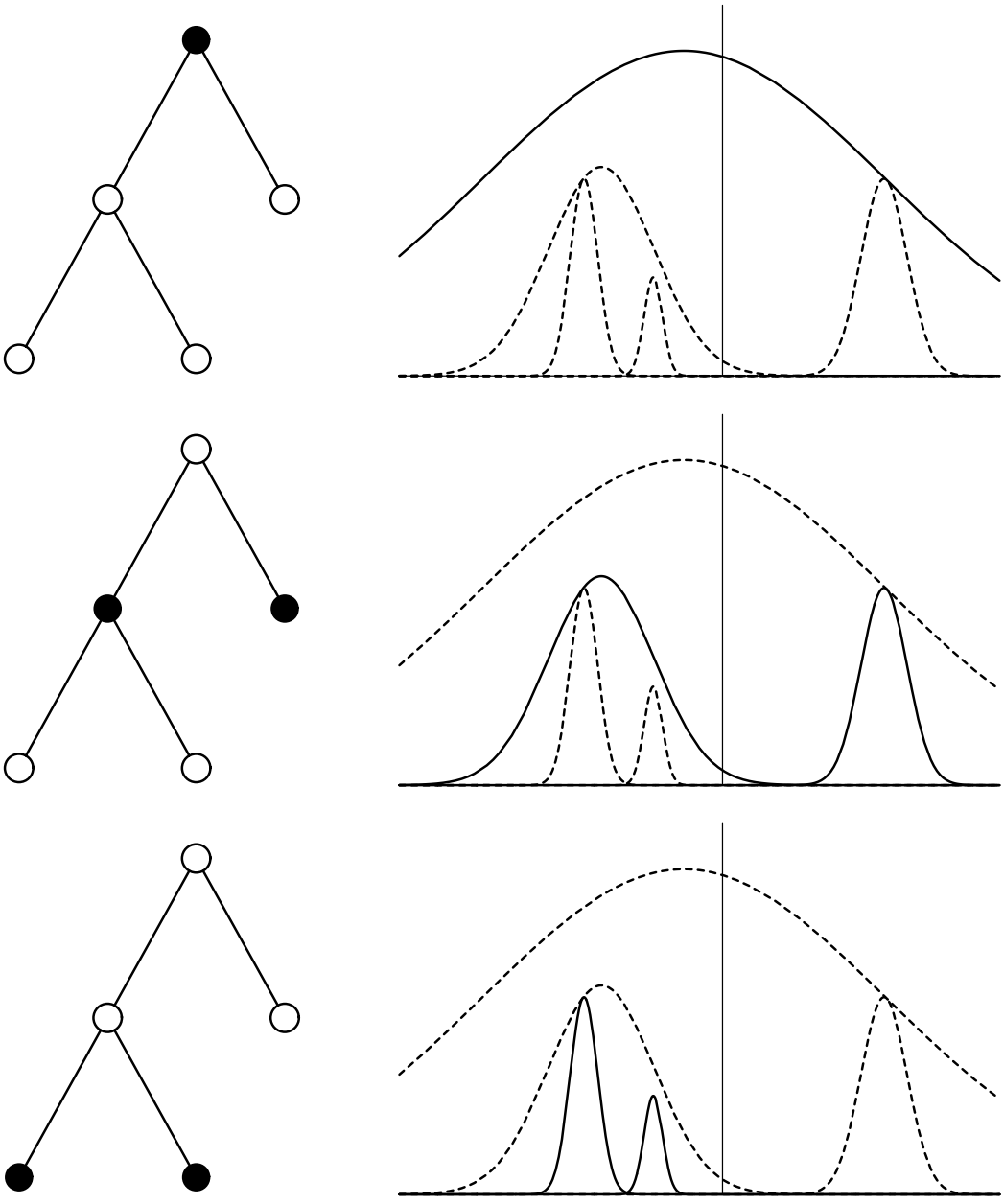

Figure 4.2: The bump functions of a parent node and its children.

tree structures is that they easily lend themselves to such branch and bound techniques. The wish for branch and bound techniques and density estimation is unified in the bumptree structure, which was briefly presented in the previous chapter. In bumptrees a node function is to be everywhere larger than its children, as seen in fig. 4.2.

In the figure, black tree nodes correspond to solid functions and white nodes to

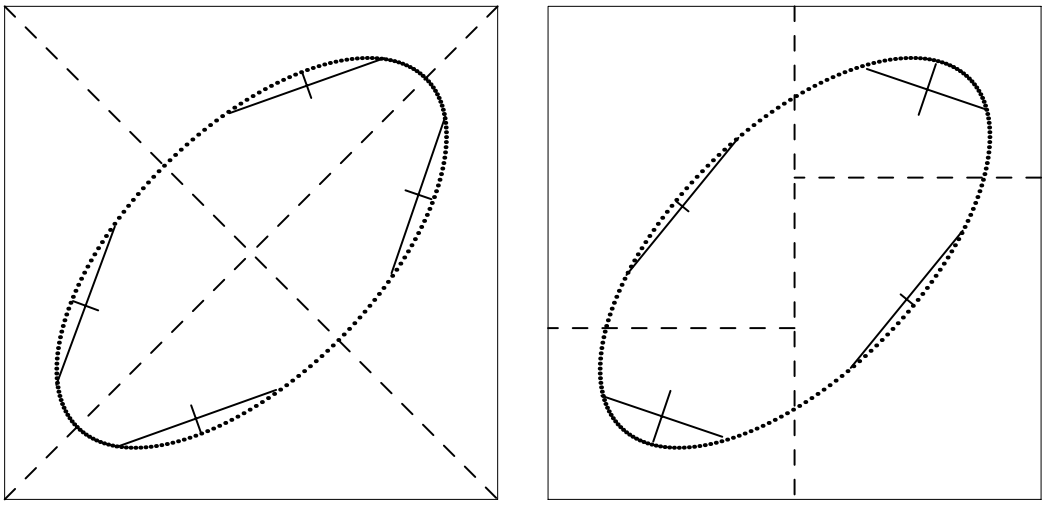

Figure 4.3: A decision distribution consisting of 200 *equally weighted samples along an ellipse partitioned by a* k-**e**-d *tree (left), and a* k-d *tree (right).*

dashed functions. The node functions in the proposed tree will be gaussians, since the distribution of responses in a region of the decision space is modeled with a normal distribution. In order to make the tree a bumptree the gaussians will have to be modified so as to be everywhere larger than their children. This is achieved by comparing the gaussians resulting from the children models to that of the parent node. The details of this procedure is will be presented in section 4.3.

When growing a k-**e**-d tree both the position and direction of a split has to be calculated, since the k-**e**-d tree allows for splits along any direction **e**, in the k-dimensional space. This is in difference with the k-d tree in which splits are bound to be along the axis. The flexibility of the k-**e**-d tree will cause the size of the tree to be kept small even if the dimensionality of the space increases. The difference, in two dimensions, is illustrated in fig. 4.3, where the crosses represent the shapes of the distributions. The k-**e**-d can be seen to provide a more adaptive partitioning of the decision distribution. Compare this difference to the one between box classification and classification using linear *discrimination functions*. The components of vectors used in classification, the *feature vectors*, correspond to a collection of features which are to separate the vectors into a number of classes. In box classification the feature space is partitioned with lines perpendicular to the feature coordinate axes, resulting in a structure identical to that of an adaptive grid described in chapter 3. Partitioning the feature space using discrimination functions bears more resemblances to the k-**e**-d structure. Each class is assigned a discrimination function which measure how probable a feature vector is to belong to the class. The boundary between two classes is given by solving for which feature vectors the discrimination functions are equal. Together with the mean vector the directional vector **e** defines a hyperplane, which is nothing but a linear boundary between two partitions. This boundary acts in the same manner as that derived from the linear discrimination functions used in

classification.

An important component in tree growing is hence to find suitable node questions, i.e. how to split the tree at each node depending on the data. This corresponds to determining the discrimination function parameters at each node. In the case of the k-**e**-d tree, this is to decide the direction, and the mean vector for the split. The common approach is to define an impurity measure and then find the split which minimizes this measure. When trees are used for classification impurity often relates to how mixed the classes are in a node. The goal is to have one leaf node for each class, meaning that the impurity is reduced from the root node towards the leaves.

Consider splitting a node representing data with a covariance matrix $\mathbf{C}$. A split results in two disjunctive sets having mean vectors $\mathbf{m}_1$, $\mathbf{m}_2$ and covariance matrices $\mathbf{C}_1$, $\mathbf{C}_2$ respectively. The total covariance of the two sets, $\mathbf{C}$, can be divided into two parts or *scatter matrices*, a matrix $\mathbf{C}_W$ representing the scatter within the sets and another measuring the scattering between the two sets, $\mathbf{C}_B$:

$$\begin{aligned} \mathbf{C} &= \mathbf{C}_W + \mathbf{C}_B \\ \mathbf{C}_W &= \frac{1}{n-1}((n_1-1)\mathbf{C}_1 + (n_2-1)\mathbf{C}_2) \\ \mathbf{C}_B &= \frac{n_1 n_2}{n(n-1)}(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \end{aligned} \tag{4.1}$$

Here n_1 and n_2 are the number of samples in the two sets caused by the split, and n equals n_1+n_2. The second term in the sum, the matrix $\mathbf{C}_B$ describing the covariance between the two sets, will provide a purity measure. Many different measures are proposed in the literature, see e.g. (Duda and Hart, 1973; Fielding, 1977). A simple one, which turns out to measure purity in terms of sparseness, is the trace of the between sets covariance matrix:

$$\eta = \mathrm{Tr}(\mathbf{C}_B) \tag{4.2}$$

This is a measure of how separated the two classes are and speaks about the variation explained by splitting a node (Fielding, 1977). Maximizing it will cause the models in the tree nodes to describe the decision space as sparsely as possible. Also notice that since

$$\mathrm{Tr}(\mathbf{C}) = \mathrm{Tr}(\mathbf{C}_W) + \mathrm{Tr}(\mathbf{C}_B) \tag{4.3}$$

is constant, maximizing $\eta = \mathrm{Tr}(\mathbf{C}_B)$ will minimize the scatter within the sets as represented by $\mathrm{Tr}(\mathbf{C}_W)$. This is compatible with the desire for a representation to be sparse and local, as discussed in the previous chapter.

The following heuristic is suggested to achieve a split which aims at maximizing the purity measure. Split the data set with the hyperplane corresponding to the largest eigenvalue of the covariance matrix $\mathbf{C}$. This results in a split through the mean vector across the direction of maximal variation and should be a proper way

to separate the two sets as much as possible. The eigenvector being normal to the hyperplane mentioned above is often referred to as the *first principal component* and is identical to the vector **e** which solves the equation

$$\mathbf{Ce} = \lambda_{max}\mathbf{e} \tag{4.4}$$

where λ_{max} is the largest eigenvalue of the covariance matrix **C**. Similar heuristics have been used for finding node questions in classification problems (Sirat and Nadal, 1990) and for the formation of new clusters in nonhierarchical clustering techniques (Ball and Hall, 1965). Also this strategy will aim at minimizing the number of subtrees needed to be investigated during the branch and bound procedure, when generating responses as mentioned in section 4.3.

Some properties of the split resulting from the above procedure are worth mentioning. One is that the split will be sensitive to linear transformations of the data. Units of measurement will hence influence the way the decision space is partitioned. This can be used if certain stimuli or responses are considered to affect the subdivision of the space more than others. If, however, all stimuli and responses are considered as equally influential, some sort of normalization procedure has to be applied to the decisions before they participate in the split calculations. Normalization should be applied with care since important spread in the data due to discontinuities and model boundaries may be lost. At the ground of this problem lies a fundamental problem from measurement theory. How should a similarity measure be designed so that decision components in various units can be compared and correlated? To answer this question is to give meaning to the learning procedure and is something the designer of the system is forced to do (Duda and Hart, 1973).

Another property is that distributions with equal 2:nd order statistics will be indistinguishable the split procedure, since it is using the first principal component only. All the distributions in fig. 4.4 share this property. If the view taken is that the total distribution at each node should be decomposed into a sum of two normal distributions this may seem a bit awkward, e.g. when looking at fig. 4.4 b.

One way to overcome this problem, in fact a way to estimate all the node parameters, would be to assume that the data after the split is partitioned into two normal distributions. Then maximum likelihood techniques could be applied to find the boundary between the two classes, and also their parameters. Maximum likelihood techniques are common and sometimes very powerful, but they will not used in this approach. Why is this? The rest of this section is supposed to answer that question.

Consider two classes, ω_1 and ω_2, modeled as normal distributions with different mean vectors and covariance matrices. The maximum likelihood discriminant functions, g_i, becomes:

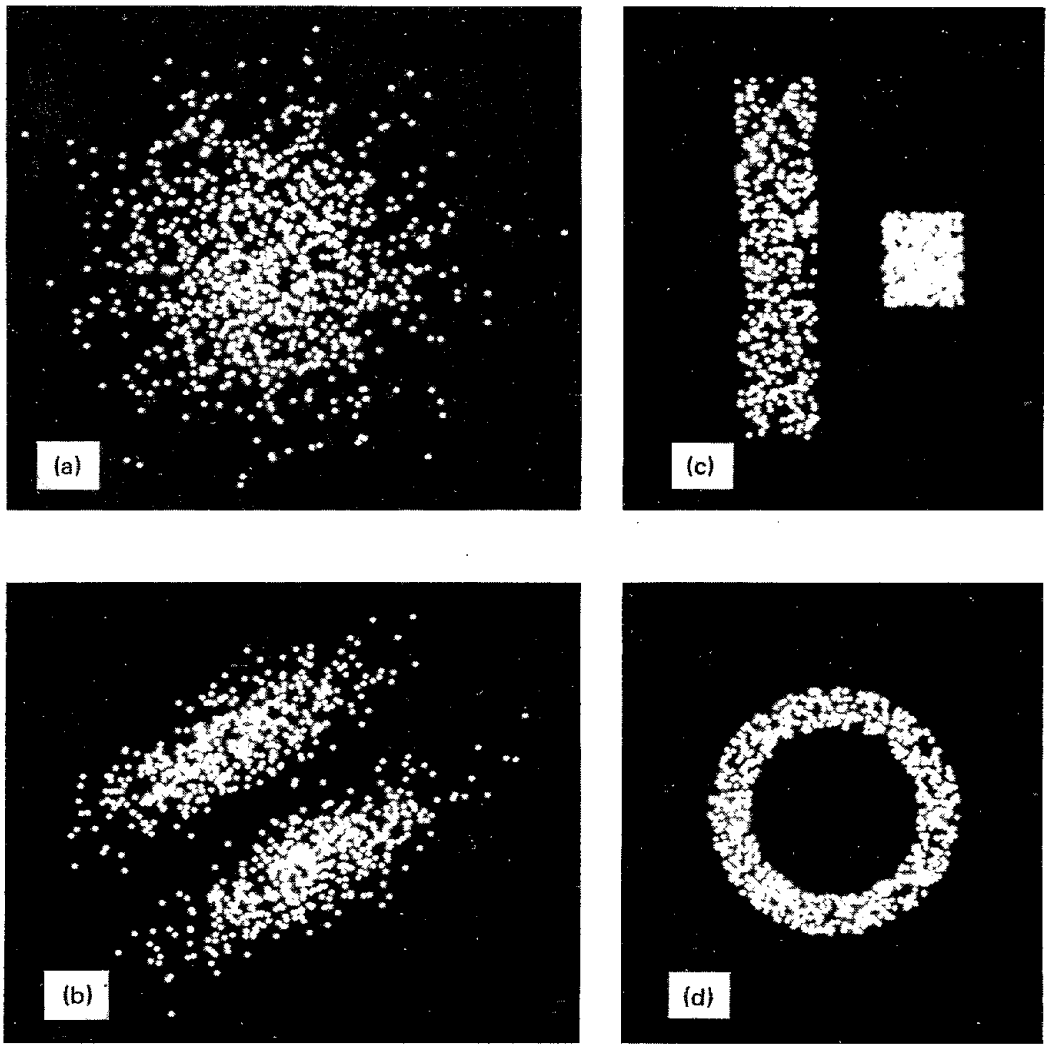

Figure 4.4: Four distributions with identical second order statistics. From (Duda and Hart, 1973).

$$
\begin{aligned}
g_i(\mathbf{x}) &= \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_{i0} && (4.5)\\
\mathbf{W}_i &= -\frac{1}{2}\mathbf{C}_i^{-1} && (4.6)\\
\mathbf{w}_i &= \mathbf{C}_i^{-1}\mathbf{m}_i && (4.7)\\
w_{i0} &= -\frac{1}{2}\mathbf{m}_i^T \mathbf{C}_i^{-1}\mathbf{m}_i - \frac{1}{2}\log|\mathbf{C}_i| + \log P(\omega_i) && (4.8)
\end{aligned}
$$

Here, $\mathbf{C}$ is the covariance matrix, $\mathbf{m}$ is the mean vector, and $P(\omega)$ is the class probability. In difference with the hyperplanar decision boundary in the k-**e**-d tree the maximum likelihood procedure results in hyperquadric surfaces. When the covariance matrices of the two distributions are identical, the maximum likelihood discriminator becomes linear, but the maximum likelihood plane will not have the

first principal component as its normal, as had the hyperplane resulting from the previously proposed split. To see this, consider two classes with equal a priori probability $P(\omega_1) = P(\omega_2)$ and identical covariance matrices, $\mathbf{C}_1 = \mathbf{C}_2 = \mathbf{C}_{12}$, separated so that the total covariance matrix describing them together becomes an identity matrix, $\mathbf{C} = \mathbf{I}$. The proposed heuristic results in a split in an arbitrary direction while a maximum likelihood procedure will result in a split according to equations eq. 4.5 – eq. 4.8:

$$(\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{C}_{12}^{-1} \mathbf{x} - (\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{C}_{12}^{-1} (\mathbf{m}_1 - \mathbf{m}_2) = 0 \tag{4.9}$$

In order to evaluate the maximum likelihood estimates, knowledge of mean vectors, covariance matrices, and class probabilities is needed. If maximum likelihood techniques are used also to estimate these three quantities, without imposing any constraints, the result in the general case will be useless singular solutions (Duda and Hart, 1973). However, restricting the attention to the largest of the finite local maxima of the likelihood function will result in meaningful solutions. To compute the mean vector, the covariance matrix, and the class probability an estimate of the conditioned probability, $P(\omega|\mathbf{x}, \mathbf{m}, \mathbf{C}, P(\omega))$, is needed. One approach would be to guess this estimate, which depends on the mean vector and the covariance matrix, and then update the estimates iteratively. The problem then is how to make the initial guess. To use the proposed heuristic split as an initial guess of $P(\omega, \cdot)$, e.g. by a variation across the guessed decision boundary, seems not to be worth the trouble. When the guess is bad there is a risk that many iterations will be needed to get it right and when the guess is a good one, further computations are unnecessary. Another argument against using the maximum likelihood approach is that the fundamental assumption, that two normal distributions will be the result of a split, in most cases will be too strong.

4.2 Growing the Tree

The two issues discussed in this section and the following one, describing the procedures for representation formation and response generation respectively, are closely related in a bootstrapping manner. To grow the tree, decision data is needed, which in turn is produced using the tree. Tree growing also bears resemblances to the clustering phase when training RBF networks and kernel estimators, as described in the previous chapter. There the number of kernels and their locations were to be estimated. Here the same problem is solved top down by partitioning the decision space till a sufficient number of kernels or models emerge. A distinction is often made between on- and off line procedures. Algorithms that continuously updates the structure are referred to as on line methods and procedures modifying structures in a batch fashion are called off line methods. The learning algorithm presented here is something of a mixture. It could be updated in every time-step but lends itself more naturally to batch processing. This is because the tree structure is grown from the root each time it is rebuilt. The experienced decisions are needed for the

estimation of child model parameters when nodes in the tree are split. If the models in the tree are updated iteratively the child models of a parent to be split has to be somehow initiated. However, if the decisions are stored, each of them can be assigned to a child node and the resulting distributions will be possible to calculate. In an iteratively built tree, a child of a split parent node may after a while experience decisions far from those described by the parent model before the split. This will cause problems since it is assumed that the parent model is the sum of the models in its children. To handle this the tree must be rebalanced from time to time, and the question of how to initiate the children arise again.

Representing the OBD means representing knowledge of what to do when performing well. A selected number of experienced decisions are stored by the system, for use in the tree growing procedure. The first-in-first-out principle, described in the previous section, is used to decide which decisions to use when growing the tree. An important feature of this algorithm is that it can benefit from the incorporation of a priori knowledge in terms of decisions known to be good. The memory buffer is simply initialized with the set of a priori decisions. Consider the effect of applying the same strategy to neural networks. Any a priory information put into the net by initialization of its weights will soon be destroyed due to the global receptive fields of the neurons. These will cause the system to alter all of its weights in order to compensate for errors.

From the beginning the system's memory is empty, or initialized with a priori knowledge. Once a specified percentage of the decisions in the systems memory are exchanged the tree is rebuilt. A new tree structure is grown by applying a recursive procedure to the stored set of experiences. The procedure of growing a tree can be summarized in five steps:

1. Calculation of the weighted mean decision vector

$$\mathbf{m} = \frac{\sum R_i \mathbf{x}_i}{\sum R_i} \tag{4.10}$$

   where $\mathbf{x}_i$ are the decisions and R_i are the weights, i.e. the reinforcements corresponding to the decisions $\mathbf{x}_i$. Without loss of generality the reinforcement is assumed to lie in the range $R \in [0, 1]$.

2. Weighted decision covariance calculation:

$$\mathbf{C}_R = \frac{\sum R_i (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T}{\sum R_i} \tag{4.11}$$

   The same line of reasoning as was presented for the calculation of the mean vector is valid for this computation.

3. Modification of the covariance matrix

   A learning system will have to balance between exploration and exploitation, as described in chapter 2. Continuity is assumed as long as it is not violated. When highly reinforced decisions are produced their information must be exploited by the system, while distributions resulting from uncertain decisions should not be allowed to introduce structure to the response generation process. If no good decisions have been experienced the system should not trust any implicit structure in the bad decisions it has experienced, see fig. 4.5. Such a structure could stem e.g. from the way the decision distribution happened to be sampled. The crosses in the figure are positioned at the weighted mean decision vectors and the crossed lines represent the eigenvectors of the covariance matrices. Line lengths are proportional to the eigenvalues. When no good model is available the system should try to explore new parts of the decision space. Such a behavior is achieved by a linear combination of the weighted covariance matrix $\mathbf{C}_R$ and an identity matrix:

$$R_m = \frac{\sum R_i \, R_i}{\sum R_i} \tag{4.12}$$

$$\mathbf{C} = w_m(R_m)\mathbf{C}_R + (1 - w_m(R_m)) \, c_E\mathbf{I} \tag{4.13}$$

   where N is the number of decisions, $\mathbf{I}$ is the identity matrix, and c_E is a measure of how explorative the system should be. The function w is a weight function which controls how the *summed* reinforcement affects the linear combination of exploitation and exploration. Again, highly reinforced decisions, which violates the expected and indicates that a model is not valid, should be able to stand out among the bad decisions and influence the calculations of a new model.

4. Calculation of the eigenvector corresponding to the largest eigenvalue of the covariance matrix $\mathbf{C}$, i.e. the first principal component.

   Since only the largest eigenvalue and its corresponding eigenvector need to be calculated it is not necessary to find the complete eigensystem solution. Instead fast iterative algorithms which only provides the largest eigenvector can be applied. Examples of such algorithms can be found e.g. in the neural network literature (Sirat and Nadal, 1990) and in textbooks on numerical analysis (Press et al., 1986; Golub and Loan, 1989). The algorithm used in the experiments is the iterative "power method" described in (Golub and Loan, 1989). This method has the complexity of a matrix multiplication and converges in a few steps.

5. Computing the inverse of the covariance matrix.

   The inverse of the covariance matrix, $\mathbf{B} = \mathbf{C}^{-1}$, will be needed in several parts of the learning algorithm. Of all calculations presented in this text, this one is the most time consuming. The inverse is stored in the corresponding

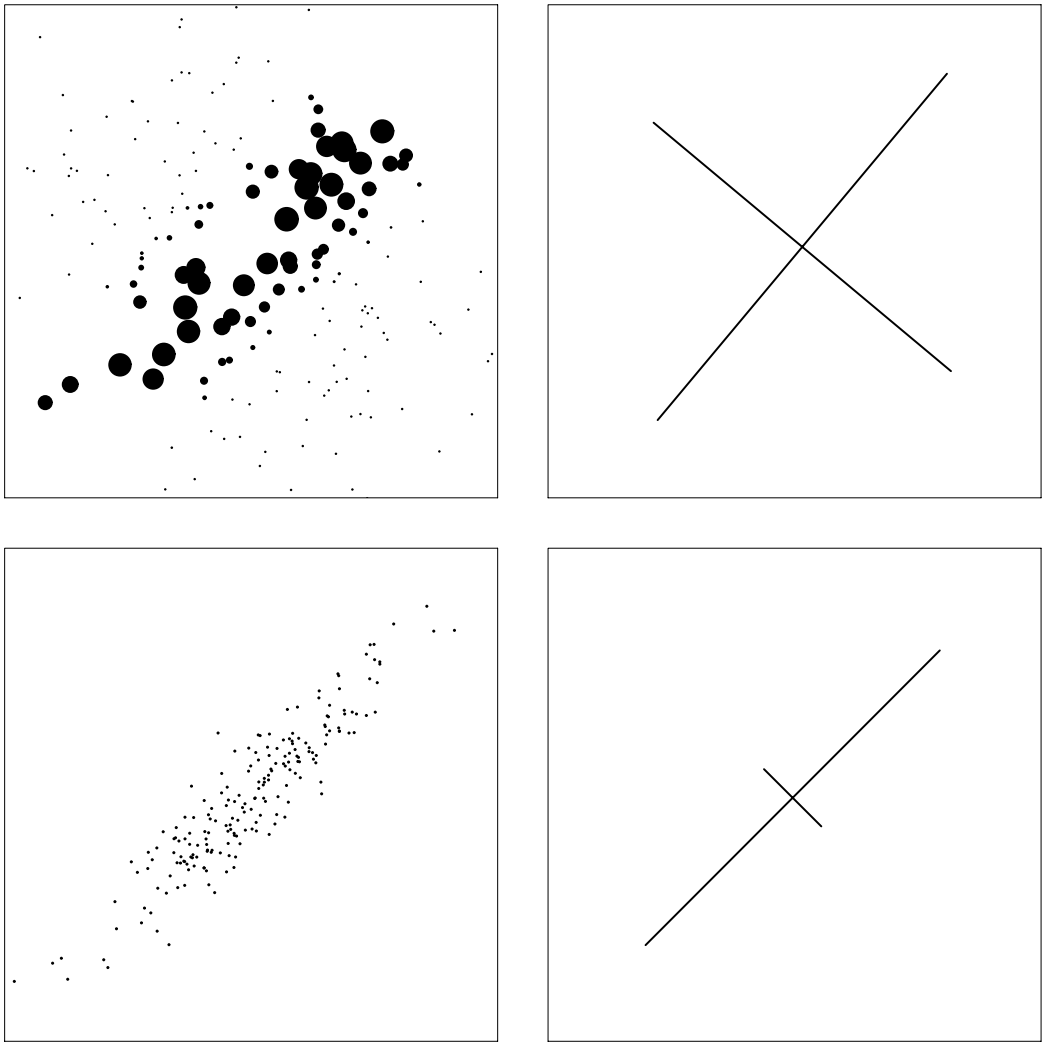

Figure 4.5: Straight forward calculation of the decision covariance matrix may result in poor estimates. Structure in highly reinforced decisions can be lost due to a large number of decisions with low reinforcement (top). Decisions with low reinforcement may impose a non-existent structure due to the sampling pattern (bottom).

node together with the mean vector, the first principal component, and the reinforcement sum.

This five step procedure is first applied to the whole set of memorized decisions and the results in terms of mean vectors, inverse covariance matrices, reinforcement sums, and eigenvectors are stored in the root node. The decision space is then divided through the mean vector into two halves by the hyperplane having the first principal component as its normal. This procedure is then repeated recursively for the data sets belonging to each of the two decision space halves, and the resulting

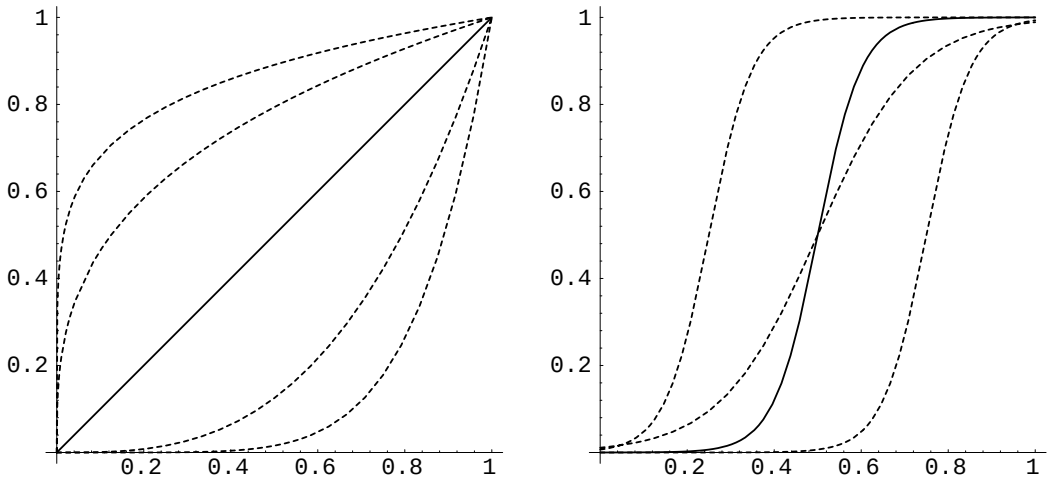

Figure 4.6: Two different families of reinforcement weight functions $w(R)$. Polynomial (left) and sigmoid like (right).

parameters are stored in the tree nodes.

The process is halted when the covariance matrix is considered to represent its decision subset well enough. In the experiments this is considered to be the case when the fraction between the largest eigenvalue and the sum of the smaller ones is large enough, and when at the same time, the mean reinforcement produced by the model is good enough. This corresponds to the assumption that the distribution of decisions is locally one-dimensional, i.e. the decisions vector traces out a curve when the system interacts with the environment according to the OBD. The behavior distribution will be adaptively partitioned since the subdivision process may be terminated at different tree depths in different parts of the tree according to the shape of the distribution. Low curvature parts of the distribution will be modeled more coarsely than highly curved parts.

The tree will represent the behavior distribution at different scales. At the top node the whole distribution is approximated with one normal distribution only. The approximation goes from coarse to fine when approaching the leaves of the tree. Also splitting through the mean vectors should cause the tree to be balanced in terms of decision probability, making each branch at the same level in the tree equally probable.

An example of a tree built from a sine wave is seen in fig. 4.7. In the left column the decomposition of the OBD into one, two, and four distributions is shown. Decision boundaries are indicated with dotted lines and the normal distributions are illustrated with lines, corresponding to the eigenvectors of the covariance matrices, originating from the mean decision vectors. The length of the eigenvector lines are in proportion to their eigenvalues. In the right column the locations of the different models in the tree are shown with filled dots. At the top right the models on all scales are shown together on top of each other.

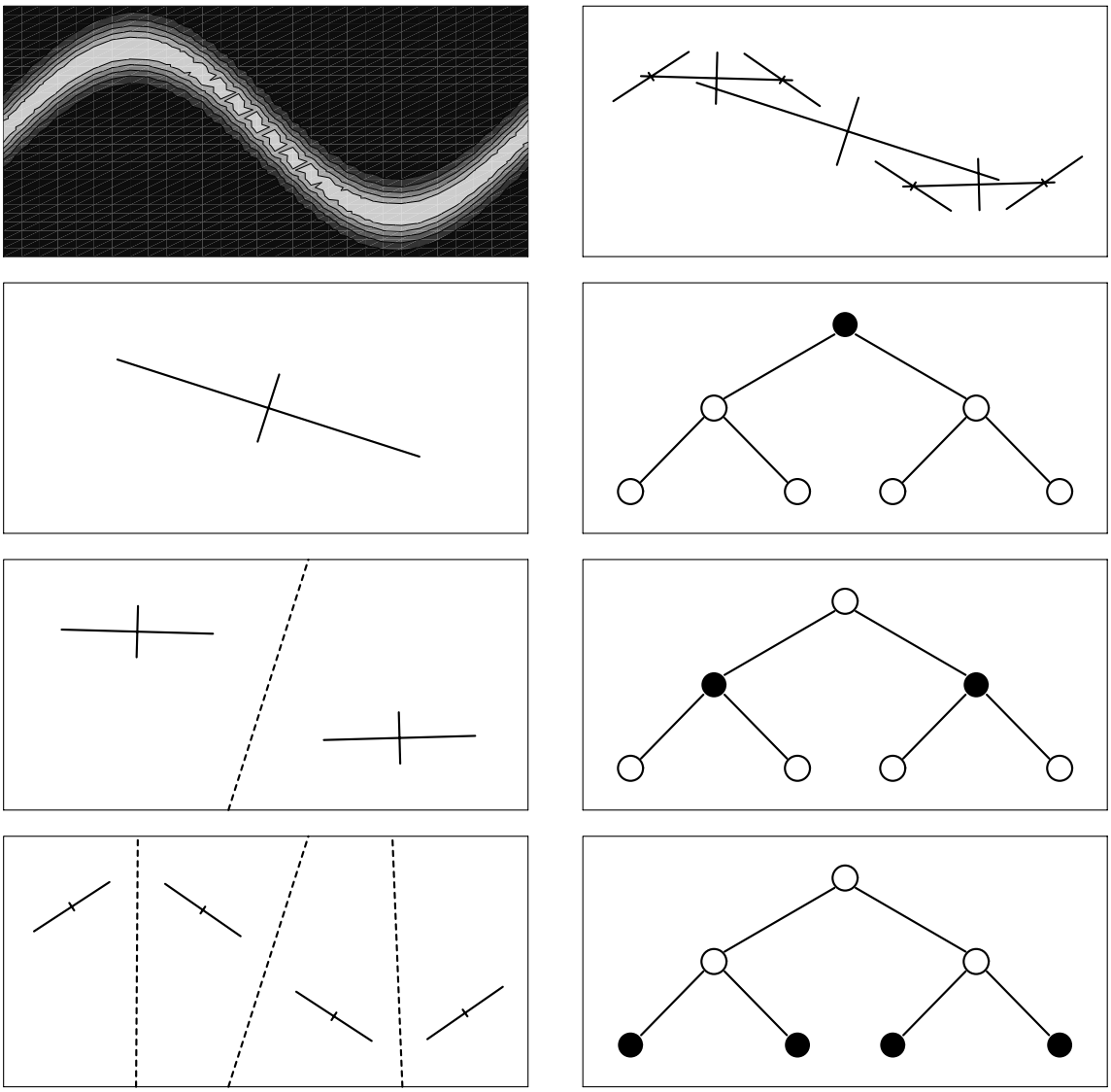

Figure 4.7: Growing a three level tree representation (top right) of a sinusoidal behavior (top left). The crosses represent normal distribution covariance matrices and are placed at the mean vectors. At the root the behavior is approximated with only one distribution. On the second and third level the distributions are split into halves, yielding two and four distributions respectively. Splits, i.e. decision boundaries, are marked with dotted lines.

4.3 Response Generation

The tree representation is built using a bootstrapping procedure where fragmented past experience, represented in the tree structure, is used for generation of new responses adding more information to the system about the environment.

Given the tree and a new stimuli, what is the system to do next? The tree is the system's guide to response generation and must be asked for responses likely to receive a high reinforcement. Each of the leaves contains a normal distribution estimating the behavior distribution in its part of the decision space. The sum of all the leaf distributions is treated as a global description $p(\mathbf{s}, \mathbf{r})$ of the behavior distribution:

$$p(\mathbf{s}, \mathbf{r}) = \sum_i \alpha_i \; p_i(\mathbf{s}, \mathbf{r}) \; , \qquad \sum_i \alpha_i = 1 \; , \; \alpha_i \geq 0 \tag{4.14}$$

Since the part of the decision vector that corresponds to the current stimulus is known a global distribution of responses likely to give high reinforcement in the current context is searched for. The solution is to use the conditioned distribution of responses $p(\mathbf{r}|\mathbf{s})$, when looking for a new response:

$$p(\mathbf{r}|\mathbf{s}) = \frac{p(\mathbf{s}, \mathbf{r})}{p(\mathbf{s})} \tag{4.15}$$

To see how this conditioned probability distribution is calculated, consider the normal distribution, $N(\mathbf{m}, \mathbf{C})$, which in k dimensions takes the form of

$$N(\mathbf{m}, \mathbf{C}) \sim p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{k}{2}} \sqrt{|\mathbf{C}|}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T \mathbf{C}^{-1} (\mathbf{x}-\mathbf{m})} \tag{4.16}$$

where $\mathbf{m}$ is the mean vector and $\mathbf{C}$ is the covariance matrix. It is shown in appendix A that the projection of a normal distribution onto the hyperplane $\mathbf{s} = \mathbf{s}_0$, specified by the current stimulus, is a new normal distribution, $p(\mathbf{r})$, times a constant:

$$p(\mathbf{s}_0, \mathbf{r}) = b(\mathbf{s}_0) \; p(\mathbf{r}) \tag{4.17}$$

In fig. 4.8 a two-dimensional normal distribution and a one-dimensional projection thereof are shown. To make the projection $p(\mathbf{s}_0, \mathbf{r})$ a valid normal distribution it has to be divided by the constant $b(\mathbf{s}_0)$ found in eq. 4.17.

The projection property of the normal distribution allows the global distribution $p(\mathbf{r}|\mathbf{s}_0)$ to be calculated as a linear combination of the conditioned leaf distributions, yielding the distribution of possible responses as a new sum of normal distributions:

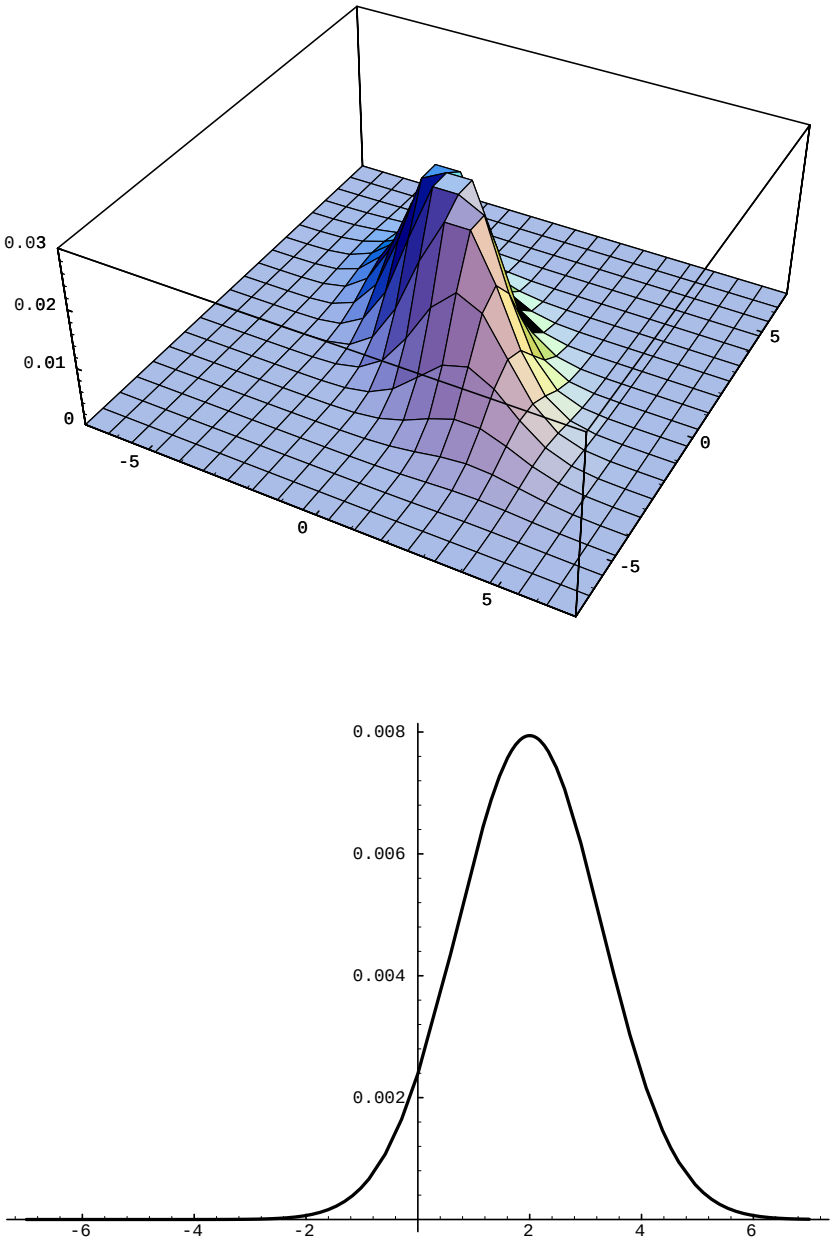

Figure 4.8: A normal distribution with zero mean and with the largest eigenvalue of its covariance matrix being five times the small one (top). A projection of the distribution along the lower axis (bottom).

$$
\begin{aligned}
p(\mathbf{r}|\mathbf{s}_0) &= \frac{1}{p(\mathbf{s}_0)} \sum_i \alpha_i \; p_i(\mathbf{s}, \mathbf{r}) \\
&= \frac{1}{p(\mathbf{s}_0)} \sum_i \alpha_i \; b_i(\mathbf{s}) \; p_i(\mathbf{r}) \quad , \; \sum_i \alpha_i = 1 \; , \; \alpha_i > 0 \\
&= \sum_i \beta_i \; p_i(\mathbf{r}) \quad , \; \sum_i \beta_i = 1 \; , \; \beta_i > 0 \qquad (4.18)
\end{aligned}
$$

Since $\sum \beta_i = 1$, where $\beta_i = \alpha_i \; b_i(\mathbf{s}_0)/p(\mathbf{s}_0)$, the probability, $p(\mathbf{s}_0)$, for the current stimulus can be computed as $\sum \alpha_i \; b_i(\mathbf{s}_0)$.

Generating a random response from the conditioned distribution in eq. 4.18 is quite easy. When a probability density function can be expressed as a linear combination

$$p(x) = \beta_1 p_1(x) + \beta_2 p_2(x) + \ldots + \beta_n p_n(x) \; , \quad \sum \beta_i = 1 \; , \quad \beta_i > 0 \tag{4.19}$$

where $p_1(x), \ldots, p_n(x)$ are probability density functions, the following two step procedure can be applied to produce a sample from the total distribution $p(x)$ (Mitrani, 1982):

1. Generate a random integer, l, being 1 with probability β_1, 2 with probability β_2 and so on.

2. Generate a random variable from the probability density function $p_l(x)$ and let it be the desired output.

The first step is achieved using a uniform random number generator and a table $a(j)$ with the accumulated sums $\sum_{i=1}^{j} \beta_i$ to compare the random number against. The table is searched for the first entry with a higher value than the random number. The index of this entry is the index l to be used in the second step. Since the distributions p_i are normal distributions, all that needs to be done is to find the parameters corresponding to the l:th distribution and feed them into a random number generator for the normal distribution. These numbers are then considered to be the final response.

Many of the coefficients in the linear combination will be small because of the global property of the distribution $p(\mathbf{r}|\mathbf{s}_0)$. Branch and bound techniques are used to prune away the leaf distributions with coefficients small enough not to influence the response generation. The coefficients, β_i in eq. 4.18, are functions of the current stimulus and correspond to the influence functions connected to bump trees described earlier in chapter 3. When generating responses the model functions are the projected versions of the leaf decision distributions. These new model and influence functions live in the response and stimuli space respectively. Note that they are not located at the response and stimulus part of the mean decision vector corresponding to the distribution from which they both originate. See appendix A and fig. 4.8.

In order to make the branch and bound procedure work, the tree has to be a bump tree. Only nodes involved in the branch and bound search need however to be transformed, since there is no need for transforming subtrees which are cut off and never visited. Note that it is also possible to use the exponent of the normal distribution when growing the bump tree. The difference between using the function itself and using its exponent will lie in what distance measure to use when deciding whether to cut a branch or not. Now consider a parent node with the model $\alpha \; p$. Its children contains the models $\alpha_1 \; p_1$ and $\alpha_2 \; p_2$ respectively. The distribution in the parent node, $\alpha \; p = (\alpha_1 + \alpha_2) \; p$, should be transformed to be a bump function, i.e. to

be everywhere larger than its children functions. A number of such transformations are possible. The simplest one is to pick the largest function value from the two children functions and let this constant be the bump function of the parent node, see the top of fig. 4.9. Such a procedure would propagate the maximum leaf function value to the root node. As a result, this node would always be considered as the one holding the most probable response model when the branch and bound procedure described earlier is used, irrespective of the current context. This problem stems from the global property of the constant functions. If replaced with box functions, being zero everywhere outside a region where the children functions are considered to be small enough, this phenomena would be avoided as illustrated in the middle of fig. 4.9. However here it is suggested that the covariance matrix is modified and that the resulting parent function is multiplied with a constant as seen at the bottom of fig. 4.9. This should reduce the risk of overestimating the leaf function values more than is the case when the constant function is replaced with a box. The first step in the modification procedure is to alter the norm of the covariance matrix and make it cover the child distribution farthest away:

$$||\mathbf{C}|| = \arg\max_k \; ||\mathbf{C}_k|| + ||\mathbf{m} - \mathbf{m}_k||^2 \tag{4.20}$$

Then the new parent distribution, with modified covariance matrix, should be multiplied with a constant c_B so that it becomes everywhere larger than its children functions for all decisions $\mathbf{x}$:

$$c_B \; (\alpha_1 + \alpha_2) \; p(\mathbf{x}) \geq \arg\max_k \; \alpha_k p_k(\mathbf{x}) \; , \quad \forall \mathbf{x} \tag{4.21}$$

The solution is presented in appendix B and yields:

$$c_B = \arg\max_k \frac{\alpha_k \sqrt{|\mathbf{C}|}}{(\alpha_1 + \alpha_2)\sqrt{|\mathbf{C}_k|}} \; e^{-\frac{1}{2}(\mathbf{m}_k - \mathbf{m})^T (\mathbf{C}_k - \mathbf{C})^{-1} (\mathbf{m}_k - \mathbf{m})} \tag{4.22}$$

When the root node function has gone through the process of bump tree transformation the search for models whose influence functions lies within a specified range from the largest one, $\beta_i > c_\beta \; \beta_{max}$, can begin. It proceeds down the most promising path, i.e down the subtree with the largest influence function value for the current stimuli. This subtree does not necessary contain the leaf with the largest coefficient since the transformation step in some cases may result in functions being larger than the node coefficients. Still it should be a good guess. The recursive search ends up at a leaf with a coefficient assumed to be the largest one and with which further comparisons may be performed. When control returns to a node, a test is made whether to also investigate its other branch or not. The search time will be affected by how often both subtrees of a node have to be investigated. The closer two node models are to each other, the greater the risk that they will be alike. This risk is reduced by the policy always to split a node at the mean decision vector and across

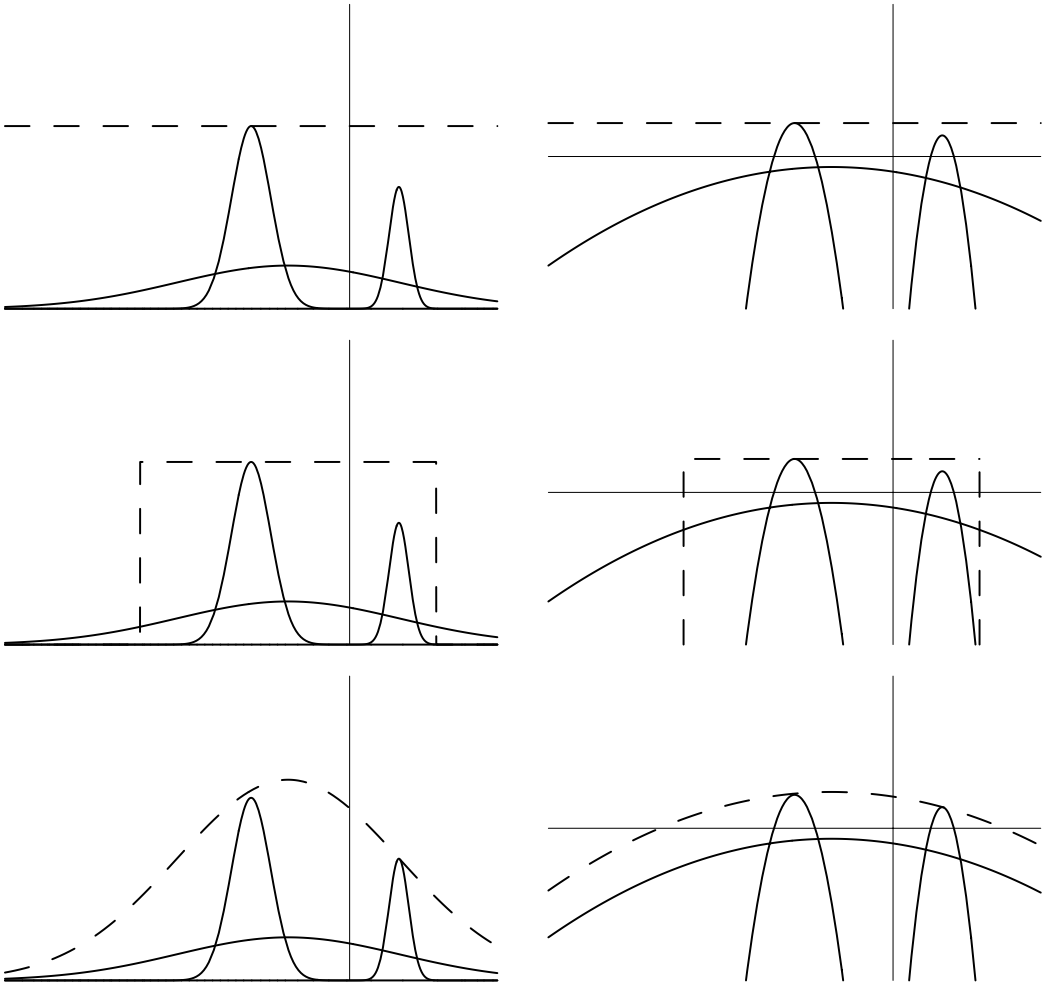

Figure 4.9: The estimated normal distributions of a parent node and its children using three different bump functions (left column). Representing the distribution with the quadric exponent (right column). The scaled parent distribution is dashed while the estimated distribution is filled.

the direction of maximal data variation. Subtrees with bump function values less than the prescribed factor times the largest coefficient found so far are pruned away and need not be further investigated. If the subtree can not be pruned away the child model is transformed into a bump function and the search continues. During the investigation of the tree a list of valid coefficients found so far is maintained. Whenever a leaf is reached its coefficient is included in the list if it is larger than the factor c_β times the largest of the stored coefficients. When all subtrees which were found worthy of investigation are searched, the list is scanned to remove coefficients which are no longer valid. At the end of this search the model list is reduced to contain only influential distributions. These are then used in the two step procedure for random response generation as described previously. This procedure leads to a

more explorative system behavior than if e.g. maximum likelihood principles were applied to the distribution $p(\mathbf{r}|\mathbf{s}_0)$. Exploration will reduce the risk of the system being trapped in a local minima and make it possible for the system to find alternative solutions when several responses are adequate to a single stimuli.

If the components of the stimuli vectors are accompanied with certainty measures it is also possible to generate responses to stimuli vectors where some of the components are lacking. Consider the case where $\dim(\mathbf{s}) = 3$ and $\dim(\mathbf{r}) = 2$, i.e $\mathbf{x} = ((s_1, s_2, s_3)^T, (r_1, r_2)^T)$. Now assuming that the component s_3 is missing in the stimuli vector, it is possible to predict it by treating the known stimuli component as a new stimuli vector $\mathbf{s}' = (s_1, s_2)^T$ and generating the response $\mathbf{r}' = (s_3, r_1, r_2)^T$. Hence the predicted value of the missing component is produced as the first component in the new response vector. This method would also be applicable to the tree growing procedure and let it make use of decision vectors where components are lacking or unsure.

Once a suitable response is found and generated the world reacts upon the decision by reinforcing it and presenting the system with a new stimulus. Reinforcing decisions is the way of telling the system when it is working well and the role of the teacher is that of constructing a critic mechanism which evaluates the overall performance of the system with a scalar value, the reinforcement signal. Designing the reinforcement mechanism is then equal to defining the problem the system is to solve since it is the only information the system has to evaluate its behavior. Once the system has received reinforcement for its decision it will face the question whether to memorize it or not. Decisions which receive low reinforcement are not remembered. If the decision was highly reinforced it is considered important only if it was generated by a model in a part of the decision space from which the system has not already received enough information. This could be checked e.g. by looking at the probability for the decision, predicted by the current tree, along with the reinforcement that the decision actually received, since the probability density $p(\mathbf{x})$ can be viewed as a reinforcement predictor.

4.4 Tree Performance

This section will deal with issues concerning error bounds and computational times for the tree structure. First the number of models needed to keep the error below a specified threshold is investigated. Then an upper bound on the number of data required to correctly estimate this number of models is presented.

The total distribution which results from the summation of the leaf models is as smooth as the local functions and the error of the sum is bounded by the errors of the leaf models. If the OBD is assumed to be concentrated to a curve in the decision space, and its curvature is bounded by a constant c_C, what can then be said about the error made when forcing it to lie along the first principal component? Such a consideration would provide a measure of how many models that are needed to produce an approximation within a given error. First look at the worst two-dimensional case where a circle

$$\mathbf{f}(t) = c_C \ (\cos(t), \sin(t))^T \tag{4.23}$$

with radius c_C, is decomposed into circle segments of length ϕ, by the k-**e**-d algorithm. Consider the case when a circle sector, without loss of generality, is positioned symmetrically around the x-axis. The circle segment is then approximated with a line, parallel to the first principal component, and passing through the centrum point, m, of the segment:

$$m = \frac{1}{\phi} \int_{-\frac{\phi}{2}}^{+\frac{\phi}{2}} c_C \ \cos\phi \ d\phi \ = \ c_C \frac{\sin \phi/2}{\phi/2} \tag{4.24}$$

This procedure will minimize the error estimated as the sum of squared distances orthogonal to the approximating line. The mean squared error for approximating a circle segment of length ϕ with a line parallel to the first principal component can then be calculated as

$$\begin{aligned} \epsilon^2 &= \frac{1}{\phi} \int_{-\frac{\phi}{2}}^{+\frac{\phi}{2}} (c_C \ \cos\phi - c_C \ \frac{\sin \phi/2}{\phi/2})^2 \ d\phi \\ &= \frac{c_C^2}{2\phi^2} \ (\phi^2 + \phi \ \sin\phi + 4\cos\phi - 4) \approx \frac{c_C^2 \phi^4}{720} \end{aligned} \tag{4.25}$$

where the last approximation step is valid for small ϕ. To produce an error which is less than ϵ, the number of models need to be of order $O(\sqrt{\frac{c_C}{\epsilon}})$, since the number of models is inversely proportional to the length of the approximation interval ϕ. With the same line of reasoning, this result could easily be extended to k dimensions by noting that the number of models, n, in this case will be inversely proportional to the interval raised to the power corresponding to the number of dimensions. However, a fundamental assumption to hold true for the k-**e**-d tree to be useful, is that the structure to be described does not vary equally fast in all dimensions simultaneously. If this assumption is valid the tree should manage to adapt to the curve as well in k dimensions as in two, with the same number of models.

Now to the question of how much data that is needed to estimate a given number of models. As described in chapter 3 a common approach to establish a bound on the worst case generalization error is to make use of the Vapnik-Chervonenkis dimension (VCD). The generalization error is meant to be the difference between the generalization on the set of memorized, highly reinforced, samples P, and the generalization on the actual task. Maybe the practical use of knowing how many highly reinforced decisions it takes to produce a tree that within some bound generates good outputs is limited, but it should at least give a hint on how much information the system need to gather before it works well. The upper bound on the VCD for an RBF network is valid also for the k-**e**-d tree, since it can be seen as a collection of RBF basis functions.

$$\text{VCD} \leq 2N_W \log(e \cdot N_N) \tag{4.26}$$

The parameter N_N denotes the number of basis functions and N_W denotes the number of weights in the linear combination of these basis functions, plus the number of parameters in the basis functions themselves. The fact that the basis functions are arranged in a hierarchical tree structure will not affect the VCD estimate, since it is only concerned with the models involved in the approximation, i.e. the ones in the leaves of the tree. Since this bound is valid only when the outputs from the hidden layer nodes are binary, a crude worst case bound for a net with real valued output could be found by assuming that each bit in the real valued output is produced by one binary net. This bound is then equal to the old one times the number of bits in the output. What does this mean for a k-**e**-d tree with n leaf models and b bits in the decision density estimate? The parameter N_N then equals n and the parameter N_W is the sum of the n coefficients in the linear combination, plus the number of parameters describing the basis functions. Since the basis functions are gaussians, each of them can be represented by a k-dimensional mean vector and a symmetric $k \times k$ covariance matrix, having $k(k+1)/2$ independent components. This line of reasoning yields:

$$N_N = n \tag{4.27}$$

$$N_W = n + n\ (k + \frac{k(k+1)}{2}) \tag{4.28}$$

which means that the upper VCD bound for a k-**e**-d tree with b bits in the output estimate can be rewritten as:

$$\text{VCD} \leq 2\ b\ n\ (1 + k + \frac{k(k+1)}{2})\ \log(e \cdot n) = O(b\ k^2 n \log n) \tag{4.29}$$

If the system in the future is to use the right model for response generation with probability $1 - \epsilon$, it will have to experience a number of highly reinforced decisions that is of order

$$O(b\ k^2 \frac{n}{\epsilon} \log \frac{n}{\epsilon}) \tag{4.30}$$

In chapter 3 it was noted that the system must have used the correct model in at least $1 - \frac{\epsilon}{2}$ percent of the previous experienced decisions for the above statement to be true.

Before any responses are generated a tree must have been grown. To get a feeling for the computational complexity of the tree growing procedure, consider the case when nodes are split in half till there is only one experience per leaf. The depth of such a perfectly balanced tree would be proportional to $\log_2 n$, where n is the total

number of experienced decisions. This can be seen as the solution to a recurrence equation

$$d(n) = d(\frac{n}{2}) + 1 \tag{4.31}$$

where the depth d of a tree is seen as the depth of a subtree plus one. At each level of the tree the inverse of the covariance matrix and the first principal component are calculated in a time proportional to k^3 and $k^2 n$ respectively. Then all n decisions, which are vectors of dimension k, has to be run through and tested for which of the two subtrees they belong to. Hence the total computational time will be proportional to $(k^3 + (k^2 + k)n) \log_2 n$, i.e. of order $O((k^3 + k^2\ n) \log_2 n)$. Again this can be viewed as the solution to a recurrence equation where the time to grow the tree is computed as the time to grow the subtrees of its two children, plus the time spent computing the matrix inverse, the first principal component, and testing the decisions on each level:

$$t(n) = 2\ t(\frac{n}{2}) + k^3 + (k^2 + k)n \tag{4.32}$$

In practise the tree will not be split till there is only one decision per leaf, but these calculations can at least be considered as a rough estimate for the complexity of growing a tree.

Once a tree is grown it will be used for generation of new responses. This process involves three steps. Branch and bound search, bump transformation, and random response generation. It can be shown that the expected number of visited leaves in the branch and bound search is independent of the number of leaf models and the probability distribution of the experienced decisions (Friedman et al., 1977). This means that the expected search time, when looking for the coefficients being within a specified range from the largest one, will be proportional to $c_L\ k^2 \log_2 n$. Here c_L is a constant representing the the number of leaves visited, and $\log_2 n$ is proportional to the depth of the tree. The factor k^2 stems from multiplications with matrices of size $k \times k$, which are involved in the bump tree transformation, performed at each level of the tree. Hence the searching time for this procedure is of order $O(k^2 \log_2 n)$. Finally the generation of a random response also require a matrix to be inverted, this time of size $m \times m$, where m is the dimensionality of the response vector. This results in a total computational time, for all three moments of response generation, which is of order $O(m^3 + k^2 \log_2 n)$. Since the dimensionality of the output will be much smaller than that of the stimulus, at least in any realistic application, the dominating term in this complexity measure will be the one involving the sum of the stimuli and response dimensions raised to the power of two. This will hold true even if the number of models does not reach towards the figure 200 mentioned in chapter 1. To see this, suppose that the dimensionality of the stimulus is a factor c_R times the dimensionality m of the response. If the quadric term is to dominate, it must be the case that $(c_R\ m + m)^2 > m^3$, which means that $m < (c_R + 1)^2$. If the dimension of the stimuli is e.g. 100 times larger than that of the response, this

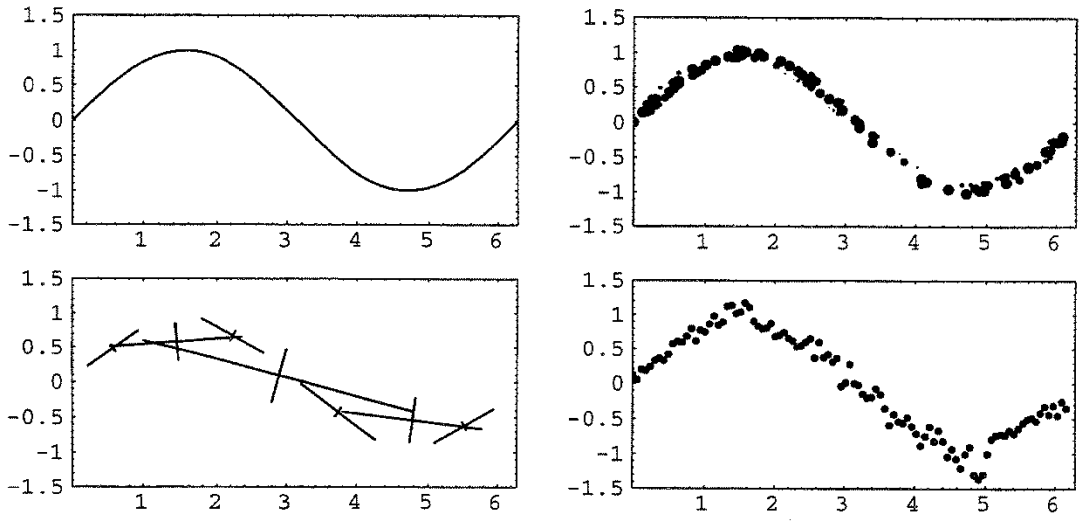

Figure 4.10: Results from the sine wave experiment.

requirement states that the dimension of the response should be smaller then 100^2, which does not seem to be an unrealistic assumption about the response. Thus, the complexity of random response generation is of order $O(k^2 \log_2 n)$. Compare this complexity to that of a fully connected two-layered MLP net, which is of order $O(kn^2)$. If the number of models, n, is assumed to be around 200 as suggested before, then the dimensionality of the decision vector need to be higher than about 5000 before the MLP net catches up on the tree. Another example is the Kohonen net, with a response generation complexity of order $O(n)$. This may look good, but it turns out to be the worst one since the number of models n happen to grow exponentially with the dimensionality k in this structure.

The complexity of the algorithm is polynomial with respect to the dimensionality of the decision space. It will however result in high computational loads when the system is to face any realistic problem, where it must cope with thousands of stimuli and response channels. Partial solutions to this problem will be proposed in section 4.6.

4.5 Experiments

Four experiments are presented. In the first two, the tree is used for one- and two-dimensional function approximation. The environment is not affected by the system responses in either of the two experiments. The third experiment, however, involves an environment responding differently to different system responses. In these three experiments the depth of the tree is fix. This is not necessary for the algorithm to converge but serves to clarify the structure of the OBD approximation. Finally in the last experiment, where the tree depth is not fix, the algorithm is tested on a problem where there are two possible good responses to some of the stimuli.

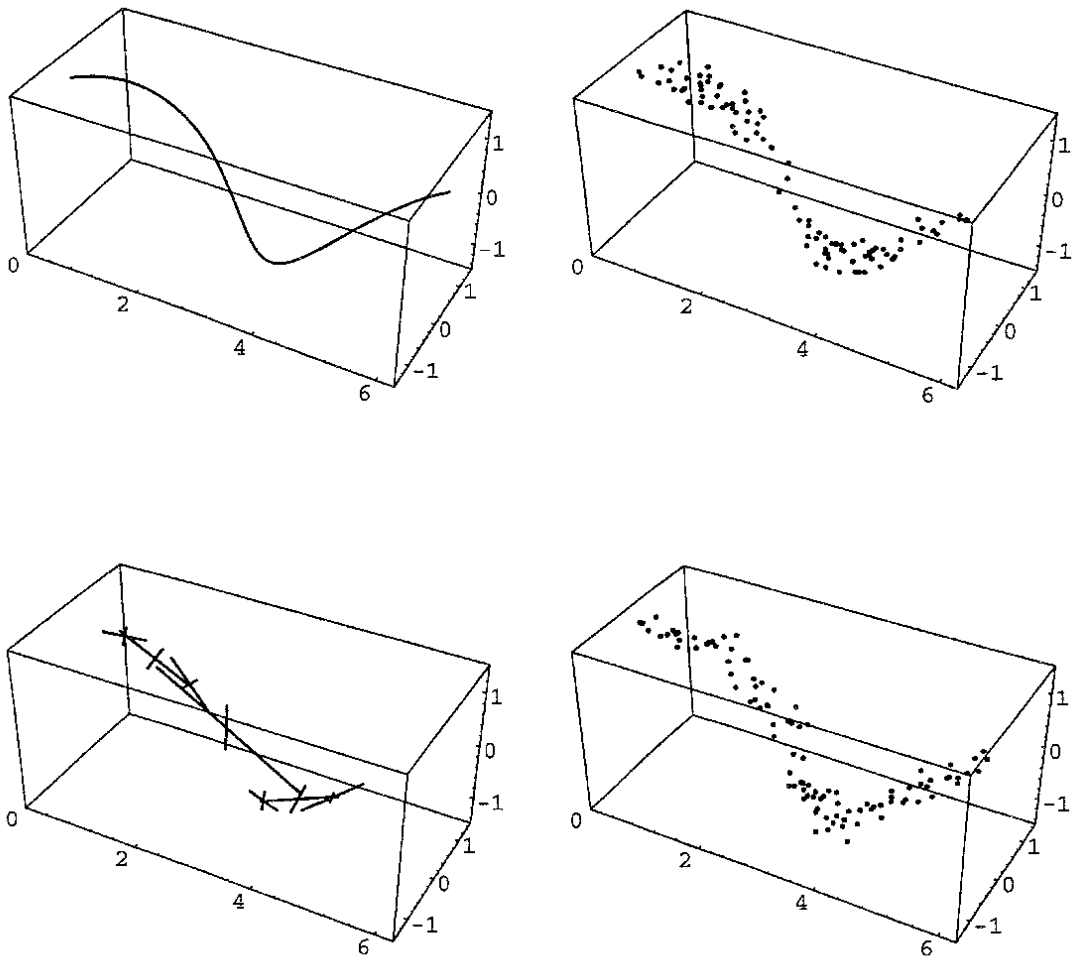

Figure 4.11: Results from the helix experiment.

Generating a Sine Wave

In this experiment the system is to learn the mapping $r = \sin(s)$, shown at the top left of fig. 4.10. The environment responds to any system response r by updating the stimuli according to $s = (s + 0.02\pi) \bmod 2\pi$ and by reinforcing the system with the scalar expression $\exp(-4(r - \sin(s))^2)$. Hence, one period of the function is sampled at 100 positions. The tree algorithm was run with the tree depth fixed to 3 and the exploration factor c_E set to 0.5. The tree was regrown whenever 10% of the stored decisions were exchanged. After 5 runs through one period of the function, the tree had the shape shown at the bottom left of fig. 4.10. The decisions that were memorized and used for tree growing are shown at the right top of the same figure. The size of the dots indicate the weight of the decisions when used in the tree growing procedure, i.e how much reinforcement the decisions received. Thereafter the tree was used for generation of one period of the sine function, without being updated. This resulted in a system behavior shown at the bottom right of fig. 4.10.

Generating a Circular Helix

This experiment is similar to the one described previously, but here the system is to learn the two-dimensional response $r = (\sin(s), \cos(s))$ to the stimuli s, as shown at the top left of fig. 4.11. The environment again responds to any system output r by updating the stimuli according to $s = (s + 0.02\pi) \bmod 2\pi$, which corresponds to a sampling frequency of 100 samples per period. Now the system is reinforced with a scalar gaussian function having its max along the helix according to the expression $\exp(-4(|r - (\sin(s), \cos(s))|)^2)$. The tree algorithm was run with the tree depth fixed to 3 and the exploration factor c_E set to 0.5. The tree was regrown whenever 10% of the memorized decisions were exchanged. After 5 runs through one period of the function, the tree had the shape shown at the bottom left of fig. 4.11. The stored decisions that were used for tree growing are shown at the right top of the same figure. This time the size of a dot is irrelevant, and the dots only indicate from which decisions the tree was grown. Using the tree for the production of one period of the function, without updating it, results in a system behavior shown at the bottom right of fig. 4.11.

Environment Interaction

An environment reacting to the response of the system is modeled in this experiment. Such a behavior of the environmental should be common feature of real life situations. The aim of the game is to add a number to the present stimulus so that the sum equals -1 when $s > 0$, and 1 whenever $s < 0$. The environment acts upon a response by adding it to the present stimulus and presenting the sum as a new stimulus to the system: $s = s + r$. Notice that this time no component in the sample position is determined a priori. System responses are reinforced with $\exp(-4(\text{sign}(s) + s + r)^2)$. The tree algorithm was run with the tree depth fixed to 2 and the exploration factor $c_E = 1$. As usual the tree was regrown whenever 10% of the saved decisions were exchanged. After 500 time steps the tree had the shape shown at the bottom left of fig. 4.12. The decisions that were stored and used for tree growing are shown at the top right of the same figure. The dot size again indicates the amount of reinforcement the decisions received. When the tree was used for response generation, without being updated for 100 time steps, the system responded according to the bottom right of fig. 4.12.

Multiple Solutions

The environment in this experiment does not react to the response of the system. What makes it interesting is that the optimal behavior of the system contains multiple solutions. A behavior similar to that resulting from this experiment would emerge if a system was to move from one point to another along a straight line, avoid an obstacle placed between the two points, and then return to the original line of motion. This would give the system the opportunity to learn two different solutions, either to pass the obstacle to the left or to the right. The system stimuli consists of its last response and the current position along the horizontal axis. The

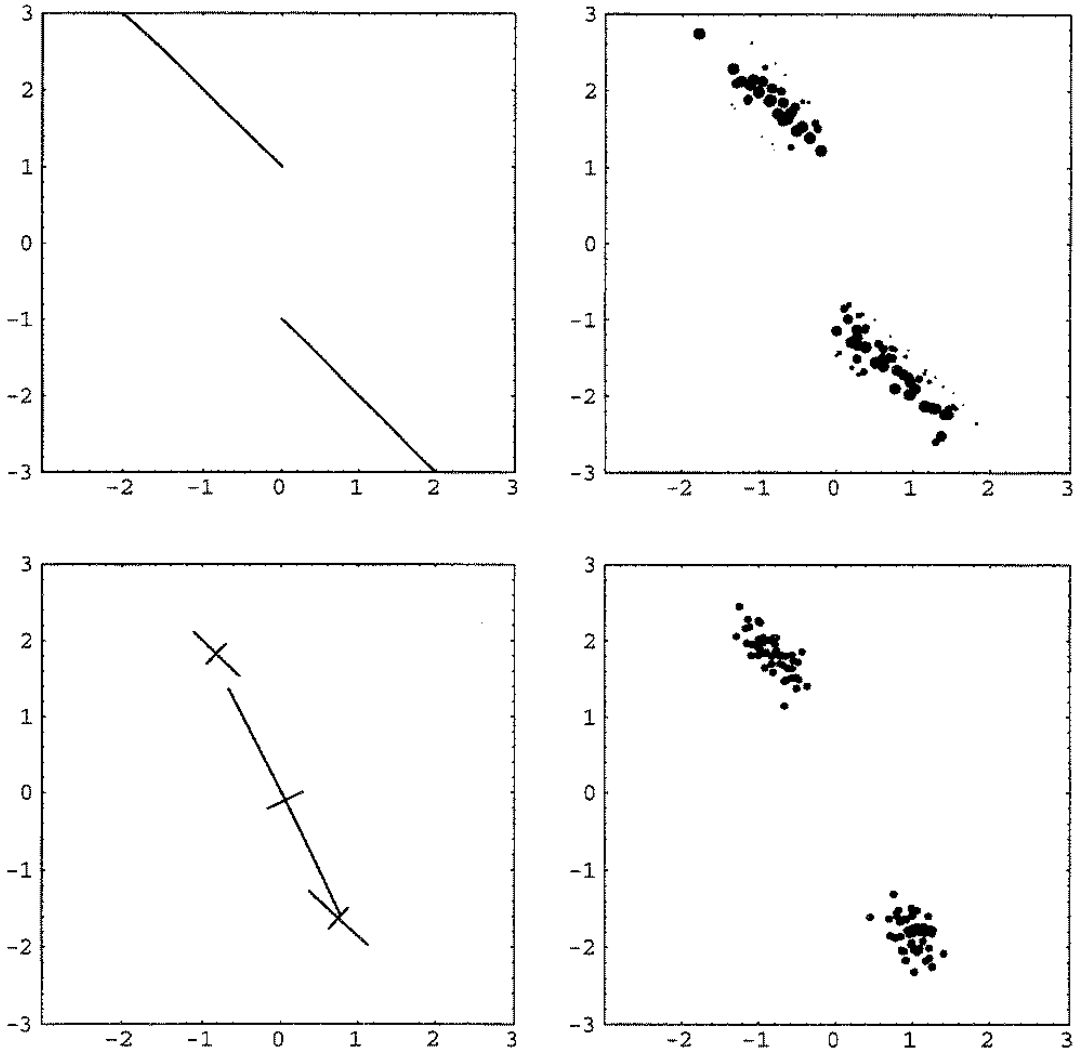

Figure 4.12: Results from the experiment with an interacting environment.

desired response is 0 until it has reached one fourth of the path length, then to either move left or right at an angle of $\frac{\pi}{4}$. When it has reached half the total length it is supposed to move back to the horizontal axis, again at the same angle. During the last part of the path the response should be the same as for the first part i.e. 0. The system was reinforced with an expression which is 1 exactly on either of the two paths and then declines exponentially with the distance to the desired paths. The depth was not fixed during this experiment but a ratio between the two largest eigenvalues, at which no further splits are imposed, was set to 0.01. To have the system explore both solutions the factor c_E was set to 25. This results in longer convergence times so this time the experiment, involving 100 samples from start to end, was run through 10 times. The decisions that grew the final tree and the final tree itself are found at the top of fig. 4.13. The tree was used a number of times for response generation along the path from start to end, without being updated.

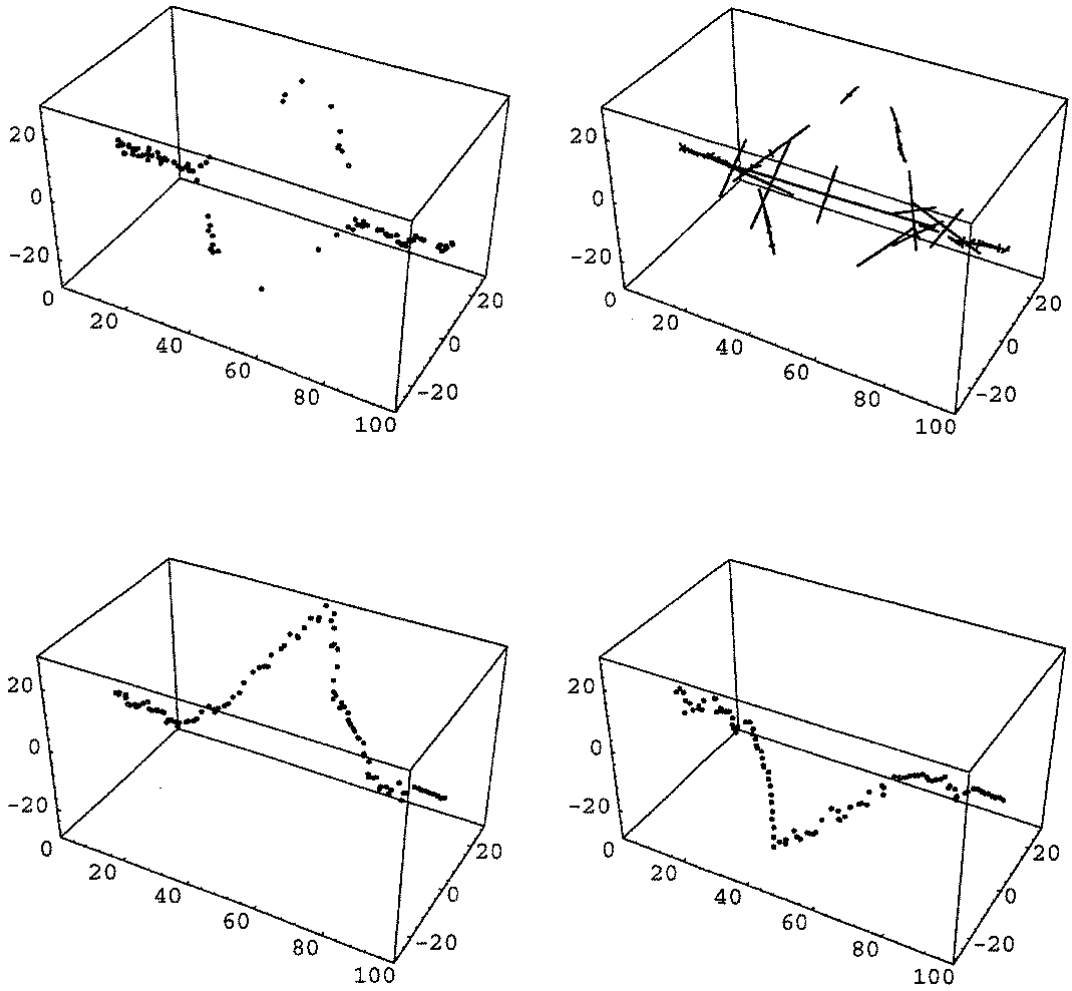

Figure 4.13: Results from the experiment with multiple solutions

This resulted in the two different system response patterns seen at the bottom of fig. 4.13.

4.6 Conclusions

The approach to model appropriate responses with a probability density function approximated by a number of normal distributions has been shown to be successful, at least on some simple yet principally important problems. The experiments show that it is possible to use the proposed binary tree structure to decompose the decision space, embedding the global behavior distribution, into smaller subsets containing one normal distribution each. The sum of these leaf distributions is then used to approximate the behavior distribution when generating responses. In reinforcement learning only a scalar value, the reinforcement, is fed back to the system as a measure of its performance. This has been verified to be informative enough for growing the

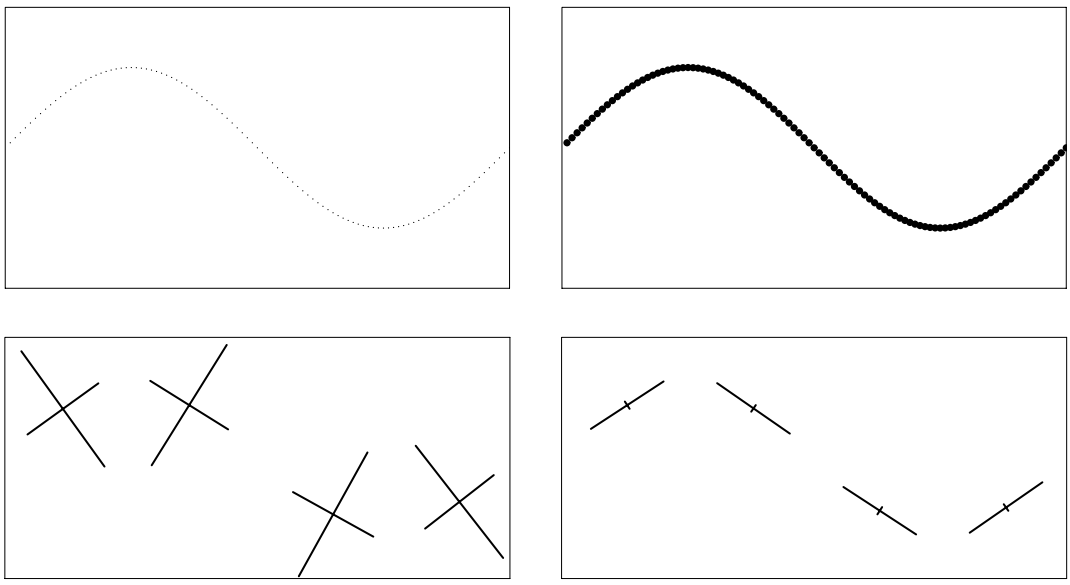

Figure 4.14: Two cases when decisions happen to be gathered along a curve. Bad (top left) and good (top right) experiences from two different tasks. The anisotropic sampling pattern is compensated for when computing the covariance matrices. Results for decisions with low reinforcement (bottom left) and for highly reinforced decisions (bottom right).

tree also in an experiment where the system was to produce two scalar outputs. In another experiment the system was able to learn two equally plausible responses to one stimulus, showing that not only function mappings are possible to represent with the tree.

There are however several points where improvements on the structure and performance of the algorithm are possible to make. Consider e.g. the way an unsure system explores the decisions space. The covariance matrix is made isotropic no matter where the previous samples where made. If the system keeps track of both the weighted covariance matrix and that of the sample points it is possible to direct the exploration towards areas which the system has not yet investigated. This should be possible to accomplish by modifying the covariance matrix $\mathbf{C}$, calculated in section 4.2. Tentatively, such a modification could be achieved by subtracting the covariance matrix, describing the samples, from the reinforcement weighted covariance matrix. When the system is very unsure of what to do, i.e. $w_m(R_m) \approx 0$, the covariance matrix to be used for generation of new responses should be, speaking sloppy, the identity matrix subtracted with the covariance matrix describing the distribution of previous samples. This means that new responses would be more probable in areas which the system has not probed so far. On the other hand if the system is sure of what to do, i.e. $w_m(R_m) \approx 1$, the modified covariance matrix becomes, again tentatively, the weighted matrix subtracted with the covariance of the samples. Now if the samples are distributed according to the good distribution described by the weighted matrix the result will be a factor times the weighted co-

variance matrix. These two cases are illustrated in fig. 4.14. The left figure shows the result when the experienced decisions are bad but happen to be concentrated along a curve. As a result, the direction of the modified covariance matrices will be opposed to that of the covariance matrices describing the samples. In the right figure highly reinforced decisions lie along the same function, which now represents the optimal behavior, and this time the modification procedure leaves the weighted covariance matrix intact.

Another point of improvement would be to find a better criteria for when to stop splitting the tree. The current criteria is based on the assumption that the underlying OBD is concentrated along a curve in the decision space. This assumption is not necessary for the rest of the algorithm to work and is rather restrictive, since there may very well be the case that hyperplanes or other multi-dimensional manifolds are considered to constitute the optimal behavior. A common view of this problem in the tree growing community is that it is impossible to find any robust and general stop criteria. Instead grow and prune strategies are suggested (Breiman et al., 1984). The problem faced when constructing stop criteria is that even though a node may seem not to benefit from being split once, it may benefit from being split twice or more. This problem could perhaps be circumvented in the special case at hand. One solution could be to look at the system performance by calculating the reinforcement mean and variance:

$$m_R = \frac{\sum R_i}{n} \tag{4.33}$$

$$\sigma_R = \frac{\sum (R_i - m_R)^2}{n-1} \tag{4.34}$$

To compute this statistics all experienced reinforcements must be considered, good as well as bad. The estimates of the mean and variance could then be used to form an Occam type of stop criteria, stating that node models which performs well, i.e. having high mean and low variance, need not be split any further. Now since a node in a bump tree structure covers the area of its children it can also be concluded that a node model which has not performed well, i.e. having low mean and variance estimates, is not likely to benefit from being divided into sub models. Finally, nodes with a large variance in their performance estimate may result in child models which adapt themselves better to the underlying distribution and hence perform better than the coarser parent model.

As stated earlier in section 4.4, a problem that must be tackled is "the curse of dimensionality". This term was coined to illustrate that system performance usually scales bad with the dimensionality of the task (Bellman, 1957). Growing the tree is of complexity $O(k^3 n log_2 n)$ due to the computation of the inverse of the covariance matrix. It is hard to see how this complexity could be reduced. The inverse is needed for response generation, and the covariance matrix itself is used when the first principal component is calculated. Regrowth of the tree is however the part of the algorithm which is invoked most infrequently, and as the system converges the need for these computations will decrease.

Moreover, if the assumption is valid which states that the OBD is concentrated along a curve, which only bends itself in a few dimensions simultaneously, then there is no need to represent the whole decision space in all levels of the tree. Instead only the subspace in which the OBD varies the most, need to be represented. Since the assumption is a local one, fewer and fewer dimensions will be needed in the models when traversing the tree towards the leaves. Thanks to the adaptability of the k-**e**-d structure to high-dimensional structures, the reduction of dimensionality should be significant even after a few splits. When making use of this principle of dimensionality reduction, the computational burden of tree growing will evidently be reduced, but still of the same order, since the dimension of the covariance matrix in the root is of size $k \times k$. The impact will be more visible in the response generating part of the algorithm, where the original complexity of order $O(m^3 + k^2 \log_2 n)$ should be possible to reduce to the order of $O(k^2 \log_2 n)$, provided that only the first principal component is needed in the leaf models.

More general fundamental aspects on how the tree could be grown and used are considered in the next chapter, which is also a brief summary of the issues discussed earlier in this text.

Chapter 5

Summary

In this final chapter the conclusions and results from the previous chapters are summed up. Also some important questions to be answered along a line of future research are identified and discussed.

5.1 Retrospective

The work presented in this text is based on the basic idea of learning by reinforcement, within the theory of behaviorism. The reason for this choice is the generality of this approach, especially that it allows the design of learning systems which can improve their behavior beyond that of their teacher. The teacher describes the problem the machine is to solve, by defining the reinforcement function. The proposed behavioristic view is to treat stimuli and responses together, as decisions. When the system is fully trained, these decisions are seen as generated from an optimal behavior distribution. This distribution is the solution to the problem the machine is about to learn. The necessity also to incorporate the responses in the organizational process has been illustrated with results from a number of psychological experiments. With a behavioristic view it is easy to motivate why this is the case. It is impossible to find any structure in the high-dimensional stimuli alone, since much of its information is totally uncorrelated with the responses a trained system is supposed to give. Meaning can only be given to the way responses appear *together* with the stimuli.

The behavior of a system, in terms of decision probabilities, can be represented in two different ways. Either let the system model the average mapping from stimuli to responses and then apply some distribution with this function as its mean value, or let the system estimate the distribution of decisions on an analytic form so that the conditioned distribution of responses given the current stimuli can be calculated once the stimuli is known. An advantage of the latter approach is that the conditioned distribution may be multimodal, which makes it possible for the system to handle cases when two or more responses are considered equally good. If as little system bias and as much flexibility as possible is preferred, the decision probability density function itself should be represented. In difference with standard neural networks which maintain a global model of its domain, the proposed tree structure models

the decision distribution locally. When the system has no or few experiences in a domain every single experience is critical. Experiences are remembered more or less in detail in the early phases, and new responses are formed by generalizing from these stored experiences. Later on, when more data is available, more complex models can be formed and there is no longer a need for using individual experiences to generate new responses. Instead the focus is on discovering regularities in the stored decisions and model them using methods for parameter fitting.

The approach to approximate the probability density function with a number of normal distributions has been shown to be successful in some simple, but principally important problems. The experiments presented in chapter 4 show that it is possible to use the proposed k-**e**-d tree structure to decompose the decision space, embedding the global behavior distribution, into smaller partitions containing one normal distribution each. The sum of these leaf distributions is then used to approximate the behavior distribution when generating responses. In reinforcement learning only a scalar value, the reinforcement, is fed back to the system as a measure of its performance. This has been verified to be enough for growing the tree structure in an experiment where the system was to produce a two-dimensional response, while only receiving a scalar error measure. In another experiment the system was able to learn two equally plausible responses to one stimulus showing that not only function mappings are possible to represent with the tree structure. As far as learning rate is concerned the proposed method seems to do better than standard neural networks. These networks typically require many presentations of *each* input only to learn a few items (Omohundro, 1987). In this reference two systems trained wih back propagation are presented. The first one solved the XOR problem when each of the four possible inputs had been presented some 500 times. The other system implemented a four bit adress decoder after the presentation of each input some 5000 times. The tree, however, needed only to try about five responses per stimulus to acquire the performance shown in the experiments.

5.2 Future Work

At the end of the previous chapter some points of improvement on the algorithm and the tree structure was presented. Here some more general aspects on where effort should be put for future development of these entities are presented. Future work should be directed towards the following five issues:

- Incorporation of temporal difference methods
- Recursive updating of the tree
- Association and generalization
- Estimation of reinforcement gradients
- Hierarchical system organization

Temporal difference (TD) methods have been used successfully in the area of reinforcement learning to deal with problems where the reinforcement is delayed or infrequent (Sutton, 1988). At the extreme a system is only made aware of success when it has completed its task. Future work will aim at incorporating the ideas of temporal difference methods into the algorithm for the learning tree. One step towards the incorporation of TD methods would be to use the tree structure for representation of $p(\mathbf{s}(t), \mathbf{r}(t), \mathbf{s}(t+1))$. Such a representation would make it possible to compute secondary reinforcement as described in chapter 2. If the system receives reward for a response it gives to a stimulus at time $t+1$ then it is possible to propagate this reinforcement back to decisions at time t, which are likely to result in an experience of the stimulus at time $t+1$. It would hence allow the system to perform a simple form of *abductive reasoning*. Abduction is the process of generating a plausible explanation for a given set of observations. This could e.g. be used to search for an explanation to how the current stimuli came to be experienced. This provides the system with what is needed in order to search for paths towards the goal by backtracking, as is done in the procedures where the method of dynamic programming is applied, which was described in chapter 2. These ideas are in line with Tollmanns theories regarding reward expectancy and goal learning, which were briefly discussed in chapter 2. As an alternative to the Watsonian view, where all the system's knowledge is based on the strengths of stimulus-response pairs formed by the received reinforcement, Tollmann proposed that the system learns goals, or to expect rewards. This means that the system learns what stimulus to expect after a given response to a certain stimulus. Stimuli associated with rewards will act as goals for the system and make it attractive to give responses that put the system in a goal state.

Instead of growing a new tree whenever it is time to update the data structure, it should be possible to update the tree structure, continuously or recursively, when new decisions worth memorizing appear. It is by no means obvious how this should be done since the partitioning of a node affects the models in its children. One way would be to build the tree one level at a time and then, when a level is considered stable, continue and graft another level of models on the tree. There are some indications that such a procedure will converge to the same tree, in terms of decision behavior, as a tree grown using the off line approach taken here (Isaksson et al., 1991). Another way to update the tree on-line was tentatively described in chapter 3 under the name of model merging. There the representation was built bottom up instead of top down as is the case with the algorithm for growing the k-**e**-d structure proposed in the previous chapter. Such a procedure should provide a good decision density estimate in less time than an approach where each level of the tree must be stable before more complex models can be grafted on a tree node.

A key to learning is the ability to associate and generalize. The system presented here has only limited abilities to generalize. As with neural nets, the generalization capability lies in the continuity of interpolation between adjacent models and of extrapolation into neighbouring areas. A more general and useful form of generalization require an ability also to associate a whole learned task with a new situation. As an illustration, consider some rats trained by Tollman to escape a maze. When

the maze was later filled with water, the rats were able to escape from the maze as fast as they did before the water was turned on. Tollman held this as an argument that it is not the individual decisions, of muscle movements to certain stimuli, that constituted the learning of the maze. Depending on at which level of abstraction stimuli and responses are connected the situation with water could be seen as a whole new task to be solved. Without the ability to generalize the new situation would require as much training of the rats as did the original task without any water. The hierarchical property of the tree structure should make induction procedures possible on different levels of abstraction, i.e. reusing old experiences from some part and level in the tree in new situations that in some way resemble each other. Which transformations of old models to consider relevant when matching them to new situations and how these are reflected in the tree structure are to be investigated. The ability to compare different subtrees with each other calls for a robust representation that does not change dramatically for small changes in the input data. A translation of the behavior distribution in the decision space will e.g. not affect the part of the tree representing the form of the distribution, i.e. the covariance matrices. Only the part of the tree representing positions will be affected, i.e. the mean vectors. This should give the tree a robustness not found in other tree representations such as e.g. k-d trees where a small change in the input data may drastically affect the resulting tree making comparisons between trees impossible.

A problem with reinforcement learning is that there is no information in the reinforcement signal itself about how to alter the behavior of the system in order to improve it. When a number of decisions have been made and reinforced it should however be possible to observe if some of them were better than others and if so, how much. The system could then make use of this implicit information about the reinforcement gradient when performing model parameter fitting. This means that also bad decisions can be helpful to the system since they together with better ones could provide an estimate of the gradient of the reinforcement signal and hence make it possible to direct exploration and estimation processes in the decision space. The main questions to be investigated related to these issues is how to robustly compute model parameters, such as mean vectors and covariance matrices, when the underlying data is incomplete an uncertain.

Many system architecture designs have been proposed throughout the years, but the mainstream one is the feed-forward input-output net. However, there are reasons to believe that the correlation between peripheral response and sensory activity is not so strong (Ballard, 1990). Biological systems seems not to solve a whole task with only one level of abstraction. Instead, it is likely that more abstract sensory and response representations are built and related to each other. This implies a learning hierarchy and that learning occurs on different temporal scales (Granlund, 1978; Granlund, 1988; Granlund and Knutsson, 1982; Granlund and Knutsson, 1990). Representing regularities in the stimuli-response information that are relevant to the behavior of the system as invariants gives the system more time to solve problems and makes it possible for the system to associate new situations with old experience. The system presented in this text could from this point of view be assigned the role of a subsystem. Many open questions in conjunction with these aspects are to be

answered, e.g. how do the subsystems emerge and connect to each other and what information is transferred between them?

From the discussion in this thesis it should be clear that data structures like the one presented, together with the paradigm of reinforcement learning, could provide an interesting alternative to present efforts towards learning systems. The research field of machine learning is only in its infancy and the work presented here is not claimed to put forward the front of the field by any considerable amount. Today a stagnation of the neural network field can be noticed. Simple and suitable problems have been solved while the generality of the approach has proved to be limited. Still, necessity is said to be the mother of inventions, which after all should inspire hope for future developements of the field. A similar description of the state of struggle was made by Churchill concerning the battle of Egypt.

> *This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.*

Appendix A

Projections of the Normal Distribution

In this appendix it is shown that a projection of a normal distribution can be written as a constant times a new normal distribution. A decision $\mathbf{x}$ can be divided into its two parts, the stimuli $\mathbf{s}$ and the response $\mathbf{r}$, $\mathbf{x} = (\mathbf{s}, \mathbf{r})$. The exponent $e(\mathbf{x})$, in a normal distribution

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{k}{2}}\sqrt{|\mathbf{C}|}}\, e^{-\frac{1}{2}e(\mathbf{x})} \tag{A.1}$$

of decisions can then, denoting the inverse of the covariance matrix $\mathbf{C}^{-1}$ with $\mathbf{B}$, be decomposed into:

$$\begin{aligned}
e(\mathbf{x}) &= (\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}) \\
&= (\mathbf{s}-\mathbf{m}_s, \mathbf{r}-\mathbf{m}_r)\mathbf{B}(\mathbf{s}-\mathbf{m}_s, \mathbf{r}-\mathbf{m}_r)^T \\
&= (u_1 \ldots u_n,\ v_1 \ldots v_m) \begin{pmatrix} \mathbf{B}_{nn} & \mathbf{B}_{nm} \\ \mathbf{B}_{mn} & \mathbf{B}_{mm} \end{pmatrix} (u_1 \ldots u_n,\ v_1 \ldots v_m)^T \\
&= (\mathbf{u}^T\mathbf{B}_{nn} + \mathbf{v}^T\mathbf{B}_{mn},\ \mathbf{u}^T\mathbf{B}_{nm} + \mathbf{v}^T\mathbf{B}_{mm})(u_1 \ldots u_n,\ v_1 \ldots v_m)^T \\
&= \mathbf{u}^T\mathbf{B}_{nn}\mathbf{u} + \mathbf{v}^T\mathbf{B}_{mn}\mathbf{u} + \mathbf{u}^T\mathbf{B}_{nm}\mathbf{v} + \mathbf{v}^T\mathbf{B}_{mm}\mathbf{v} \\
&= \left\{\mathbf{C} = \mathbf{C}^T \Rightarrow \mathbf{B} = \mathbf{B}^T\right\} \\
&= \mathbf{u}^T\mathbf{B}_{nn}\mathbf{u} + 2\mathbf{u}^T\mathbf{B}_{nm}\mathbf{v} + \mathbf{v}^T\mathbf{B}_{mm}\mathbf{v}
\end{aligned} \tag{A.2}$$

The normal distribution expressed in terms of $\mathbf{u} = \mathbf{s} - \mathbf{m}_s$ and $\mathbf{v} = \mathbf{r} - \mathbf{m}_r$ then becomes

$$p(\mathbf{s}, \mathbf{r}) = \frac{1}{(2\pi)^{\frac{k}{2}}\sqrt{|\mathbf{C}|}}\, e^{-\frac{1}{2}(\mathbf{u}^T\mathbf{B}_{nn}\mathbf{u} + 2\mathbf{u}^T\mathbf{B}_{nm}\mathbf{v} + \mathbf{v}^T\mathbf{B}_{mm}\mathbf{v})} \tag{A.3}$$

Since the stimuli $\mathbf{s}$, and and the inverse, $\mathbf{B}$, of the covariance matrix are known, this expression can be viewed as a constant times a normal distribution $p(\mathbf{r})$, of the response $\mathbf{r}$:

$$p(\mathbf{s}, \mathbf{r}) = b(\mathbf{s})\ p(\mathbf{r}) \tag{A.4}$$

To see this, rewrite the exponent in the decision distribution, $p(\mathbf{s}, \mathbf{r})$ of eq. A.3 as:

$$\mathbf{u}^T\mathbf{B}_{nn}\mathbf{u} + 2\mathbf{u}^T\mathbf{B}_{nm}\mathbf{v} + \mathbf{v}^T\mathbf{B}_{mm}\mathbf{v} = (\mathbf{v}+\mathbf{w})^T\mathbf{B}_{mm}(\mathbf{v}+\mathbf{w}) + l(\mathbf{u}) \tag{A.5}$$

Identify the terms where $\mathbf{v}$ and $\mathbf{w}$ are mixed so that eq. A.5 holds true.

$$\begin{aligned} \mathbf{w}^T\mathbf{B}_{mm}\mathbf{v} = \mathbf{u}^T\mathbf{B}_{nm}\mathbf{v} \quad &\Rightarrow \quad \mathbf{w} = \mathbf{B}_{mm}^{-1}\mathbf{B}_{mn}\mathbf{u} \\ &\Rightarrow \quad l(\mathbf{u}) = \mathbf{u}^T(\mathbf{B}_{nn} - \mathbf{B}_{nm}\mathbf{B}_{mm}^{-1}\mathbf{B}_{mn})\mathbf{u} \end{aligned} \tag{A.6}$$

Now let $\mathbf{v} + \mathbf{w} = \mathbf{r} - \mathbf{m}_w$. The new mean vector $\mathbf{m}_w$ in the normal distribution of $\mathbf{r}$ will then become

$$\mathbf{m}_w = \mathbf{m}_r - \mathbf{B}_{mm}^{-1}\mathbf{B}_{mn}(\mathbf{s} - \mathbf{m}_s) \tag{A.7}$$

The final distribution of responses then takes the following form as a normal distribution times a constant:

$$p(\mathbf{s}, \mathbf{r}) = b(\mathbf{s})\ p(\mathbf{r}) \tag{A.8}$$

where

$$b(\mathbf{s}) = \frac{\sqrt{|\mathbf{B}_{mm}^{-1}|}}{(2\pi)^{\frac{n}{2}}\sqrt{|\mathbf{C}|}}\ e^{-\frac{1}{2}(\mathbf{s}-\mathbf{m}_s)^T(\mathbf{B}_{nn}-\mathbf{B}_{nm}\mathbf{B}_{mm}^{-1}\mathbf{B}_{mn})(\mathbf{s}-\mathbf{m}_s)} \tag{A.9}$$

$$p(\mathbf{r}) = \frac{1}{(2\pi)^{\frac{m}{2}}\sqrt{|\mathbf{B}_{mm}^{-1}|}}\ e^{-\frac{1}{2}(\mathbf{r}-\mathbf{m}_w)^T\mathbf{B}_{mm}(\mathbf{r}-\mathbf{m}_w)} \tag{A.10}$$

Appendix B

Bump Tree Transformations

The requirement on a bump tree is that the parent function is everywhere larger than the functions of its children:

$$c_B \ (\alpha_1 + \alpha_2) \ p(\mathbf{x}) \geq \arg\max_k \ \alpha_k p_k(\mathbf{x}) \ , \quad \forall \mathbf{x} \tag{B.1}$$

Let M be the index of the child function which maximizes its difference to the parent function and hence the multiplicative constant c_B. Since the probability functions are normals the requirement can be restated as:

$$c_B \frac{\alpha_1 + \alpha_2}{(2\pi)^{\frac{k}{2}}\sqrt{|\mathbf{C}|}} \ e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})} \geq \frac{\alpha_M}{(2\pi)^{\frac{k}{2}}\sqrt{|\mathbf{C}_M|}} \ e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_M)^T\mathbf{C}_M^{-1}(\mathbf{x}-\mathbf{m}_M)} \tag{B.2}$$

Taking the natural logarithm of both sides of the inequality and denoting the inverse of a covariance matrix, $\mathbf{C}^{-1}$, with $\mathbf{B}$ yields:

$$\begin{aligned}
(\mathbf{x}-\mathbf{m})^T\mathbf{B}(\mathbf{x}-\mathbf{m}) + (\mathbf{x}-\mathbf{m}_M)^T\mathbf{C}_M(\mathbf{x}-\mathbf{m}_M) &\geq \gamma && \text{(B.3)}\\
\mathbf{x}^T(\mathbf{B}_M - \mathbf{B})\mathbf{x} + \mathbf{x}^T(\mathbf{B}\mathbf{m} - \mathbf{B}_M\mathbf{m}_M) + & && \text{(B.4)}\\
(\mathbf{m}_M^T\mathbf{B}_M\mathbf{m}_M - \mathbf{m}^T\mathbf{B}\mathbf{m}) &\geq \gamma && \text{(B.5)}\\
h(\mathbf{x}) &\geq \gamma && \text{(B.6)}
\end{aligned}$$

where

$$\gamma = 2 \ \ln \frac{\alpha_M|\mathbf{C}|}{c_B(\alpha_1+\alpha_2)|\mathbf{C}_M|} \tag{B.7}$$

Differentiating the expression $h(\mathbf{x})$ in eq. B.6 with respect to each component in the decision vector $\mathbf{x}$ and setting them to zero leads to:

$$\begin{aligned}
(\mathbf{B}_M - \mathbf{B})\,\mathbf{x} &= \mathbf{B}_M\mathbf{m}_M - \mathbf{B}\mathbf{m} && \text{(B.8)}\\
\mathbf{x} &= (\mathbf{B}_M - \mathbf{B})^{-1}(\mathbf{B}_M\mathbf{m}_M - \mathbf{B}\mathbf{m}) && \text{(B.9)}
\end{aligned}$$

Inserting the expression for $\mathbf{x}$ into eq. B.6, expanding the second order terms and using that the inverse of a symmetric covariance matrix also is symmetric yields:

$$(\mathbf{B}_M\mathbf{m}_M - \mathbf{B}\mathbf{m})^T(\mathbf{B}_M - \mathbf{B})^{-1}(\mathbf{B}_M\mathbf{m}_M - \mathbf{B}\mathbf{m}) + (\mathbf{m}_M^T\mathbf{B}_M\mathbf{m}_M - \mathbf{m}^T\mathbf{B}\mathbf{m}) \geq \gamma \tag{B.10}$$

This equation can be simplified using the following expression for the inverse of the sum of two matrices found in (Kailath, 1980):

$$(\mathbf{A} + \mathbf{B})^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}(\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}\mathbf{A}^{-1} \tag{B.11}$$

Using this expression turns eq. B.10 into the following simpler equation, now back in terms of covariance matrices:

$$(\mathbf{m}_M - \mathbf{m})^T(\mathbf{C}_M - \mathbf{C})^{-1}(\mathbf{m}_M - \mathbf{m}) + \tag{B.12}$$

$$\mathbf{m}_M^T(\mathbf{C}_M - \mathbf{C})^{-1}\mathbf{m} - \mathbf{m}_M^T(\mathbf{C}_M(\mathbf{C}_M^{-1} - \mathbf{C}^{-1})\mathbf{C})^{-1}\mathbf{m} \geq \gamma \tag{B.13}$$

$$(\mathbf{m}_M - \mathbf{m})^T(\mathbf{C}_M - \mathbf{C})^{-1}(\mathbf{m}_M - \mathbf{m}) \geq \gamma \tag{B.14}$$

And finally the resulting expression for the multiplication constant is given by inserting the expression for γ in eq. B.7 into eq. B.14 and solving for c_B:

$$c_B \geq \frac{\alpha_M\sqrt{|\mathbf{C}|}}{(\alpha_1 + \alpha_2)\sqrt{|\mathbf{C}_M|}}\, e^{-\frac{1}{2}(\mathbf{m}_M - \mathbf{m})^T(\mathbf{C}_M - \mathbf{C})^{-1}(\mathbf{m}_M - \mathbf{m})} \tag{B.15}$$

Appendix C

Notations

This appendix presents the notations used in the thesis. First some general guidelines. Lowercase letters are used for scalars, lower case letters in boldface denote vectors, and uppercase boldface letters represent matrices. For symbols with several meanings, it is stated in the text when to use which one.

Functions and operators

$[\cdot]$	Floor operator.
$\bigcup$	Union operator.
$\setminus$	Set minus operator.
$G(\cdot)$	Generalization measure.
$N(\cdot, \cdot)$	Normal distribution.
$O(\cdot)$	Big ordo.
$P(\cdot)$	Probability measure.
$R(\cdot)$	Received reinforcement.
$d(\cdot)$	Response predictor or depth of a tree.
$f(\cdot)$	Scalar function.
$g(\cdot)$	Probability distribution or discriminant function.
$\log(\cdot)$	The natural logarithm.
$\log_2(\cdot)$	Base two logarithm.
$p(\cdot),\ p_i(\cdot)$	Probability distributions.
$w(\cdot)$	Weight or influence functions.
$\mathbf{f}(\cdot)$	Vector function.
$\mathbf{m}(\cdot)$	Mean vector function.
$\hat{\mathbf{x}}$	Prediction of a vector.
$\mathbf{x}^T,\ \mathbf{X}^T$	Transpose of vector or matrix.
$\lvert\mathbf{X}\rvert$	Matrix determinant or set cardinality.
$\lvert\lvert\mathbf{x}\rvert\rvert,\ \lvert\lvert\mathbf{X}\rvert\rvert$	Norm of vector or matrix.
$\mathrm{Tr}(\mathbf{X})$	Trace of matrix.

Upper case symbols

D	Space of decisions.
N	Number of something.
P	Set of decisions.
Q	Set of functions.
R	Space of responses.
R_n	Accumulated reward.
S	Space of stimuli or set of samples.

Lowercase symbols

α	Weight in linear combinations and sums.
β	Probability weight coefficient.
ϵ	Error measure.
η	Purity measure.
λ	Eigenvalue.
$\mu,\ \mu_i$	Mean parameters.
$\sigma,\ \sigma_s,\ \sigma_r$	Variance parameters.
ϕ	Angle parameter.
ω	Class.
a	Table with accumulated sums of random integers.
b	Number of bits in a real valued output parameter.
c_β	Influence constant used in branch and bound search.
c_B	Constant used in the bumptree modification.
c_C	Upper bound of curvature.
c_E	Constant for how explorative a system is.
c_L	Number of leaves visited in branch and bound search.
c_R	Constant relating stimuli and response dimensionality.
d	Specific dimension or component of address vector.
f_k	Function components.
i	Index in sums and vectors.
j	Index in sums and to bit positions.
k	Index or number of dimensions.
l	Index or random integer.
m	Mean value, centrum point, or dimensionality of the response.
n	Number or size of something.
p_i	Function parameter.
q	Specific function in a set of functions.
$r,\ r_i$	Scalar response or component of response vector.
$s,\ s_i$	Scalar stimulus or component of stimulus vector.
t	Time parameter.
v	Number of subsets.

Boldface uppercase symbols

B Inverse of the covariance matrix.
C Covariance matrix.
I Identity matrix.
W Weight matrix.

Boldface lowercase symbols

e Eigenvector.
m Mean vector.
r Response vector.
s Stimulus vector.
u Input vector to the output layer in an MLP.
w Weight vector.
x Decision vector.

Bibliography

Azrin, N. H. and Holz, W. C. (1966). *Operant behavior: Areas of research and application*, chapter Punishment. Appleton-Century-Crofts, New York. W. K. Honig, Eds.

Ball, G. H. and Hall, D. J. (1965). Isodata, a novel method of data analysis and pattern classification. Report, Stanford Research Institute.

Ballard, D. H. (1990). *Computational Neuroscience*, chapter Modular Learning in Hierarchical Neural Networks. MIT Press. E. L. Schwartz, Ed.

Barto, A. G. (1992). *Handbook of Intelligent Control*, chapter Reinforcement Learning and Adaptive Critic Methods. Van Nostrand Reinhold, New York. D. A. White and D. A. Sofge, Eds.

Barto, A. G. and Singh, S. P. (1990). On the computational economics of reinforcemnent learning. In et. al., D. S. T., editor, *Connectionist Models, Proc. of the 1990 Summer School*, pages 35–44, San Mateo, CA. Morgan Kaufmann.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(8):834–846.

Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization. *Neural Computation*, 1:151–160.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, N.J.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, 24:377–380.

Bower, G. H. and Hilgard, E. R. (1981). *Theories of Learning*. Prentice–Hall, Englewood Cliffs, N.J. 07632, 5 edition.

Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems 5*, San Mateo, CA. Morgan Kaufmann.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. The Wadsworth Statistics Probability Series. Wadsworth & Brooks, Monterey, California.

Buntine, W. (1990). Myths and legends in learning classification rules. In *AAAI-90 Proceedings. Eight National Conference on Artificial Intelligence*, pages 736–742.

Cybenko, G. (1989). Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

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

Ferster, C. S. and Skinner, B. F. (1957). *Schedules of reinforcement*. Appleton-Century-Crofts, New York.

Fielding, A. (1977). Binary segmentation: the automatic interaction detector and related techniques for exploring data structure. In O'Muircheartaigh, C. A. and Payne, C., editors, *The analysis of survey data*, volume 1. Chichester: Wiley.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

Friedman, J. H. (1979). *Smoothing techniques for curve estimation*, chapter A tree-structured approach to nonparametric multiple regression. Springer Verlag, Berlin. T. Gasser and M. Rosenblatt, Eds.

Friedman, J. H., j. L. Bentley, and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software*, 3:209–226.

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *J. Amer. Stat. Assoc.*, 76:817–823.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58.

Golub, G. H. and Loan, C. F. V. (1989). *Matrix Computations*. The Johns Hopkins University Press, second edition.

Granlund, G. H. (1978). In search of a general picture processing operator. *Computer Graphics and Image Processing*, 8(2):155–178.

Granlund, G. H. (1988). Integrated analysis-response structures for robotics systems. Report LiTH–ISY–I–0932, Computer Vision Laboratory, Linköping University, Sweden.

Granlund, G. H. and Knutsson, H. (1982). Hierarchical processing of structural information in artificial intelligence. In *Proceedings of 1982 IEEE Conference on Acoustics, Speech and Signal Processing*, Paris. IEEE. Invited Paper.

Granlund, G. H. and Knutsson, H. (1983). Contrast of structured and homogenous representations. In Braddick, O. J. and Sleigh, A. C., editors, *Physical and Biological Processing of Images*, pages 282–303. Springer Verlag, Berlin.

Granlund, G. H. and Knutsson, H. (1990). Compact associative representation of visual information. In *Proceedings of The 10th International Conference on Pattern Recognition*. Report LiTH–ISY–I–1091, Linköping University, Sweden, 1990.

Guthrie, E. R. (1935). *The psychology of learning*. Harper & Row, New York.

Hajnal, A., Maass, W., Pudlak, P., Szegedy, M., and Turan, G. (1987). Threshold circuits of bounded depth. In *Proceedings of the 1987 IEEE Symposium on the Foundations of Computer Science*, pages 99–110.

Held, R. and Bossom, J. (1961). Neonatal deprivation and adult rearrangement. Complementary techniques for analyzing plastic sensory–motor coordinations. *Journal of Comparative and Physiological Psychology*, pages 33–37.

Held, R. and Hein, A. (1958). Adaptation of disarranged hand–eye coordination contingent upon re-afferent stimulation. *Perceptual and Motor Skills*, (8):87–90.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational capabilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

Hush, D. R. and Horne, B. G. (1993). Progress in supervised neural networks. *IEEE Signal Processing magazine*, pages 8–39.

Isaksson, A. J., Ljung, L., and Strömberg, J.-E. (1991). On recursive construction of trees as models of dynamical systems. report LiTH–ISY–1246, Department of Electrical Engineering, Linköping University, Sweden.

Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, MA.

Kailath, T. (1980). *Linear Systems*. Information and System Sciences Series. Prentice-Hall, Englewood Cliffs, N.J.

Kant, I. (1952). *The critique of pure reason*, volume 42 of *Great books of the Western World*. Chicago Encyclopdia Britannica cop. First published in German in 1781.

Kimble, G. A. (1961). *Hilgard and Marquis' conditioning and learning*. Appleton-Century-Crofts, New York, 2nd edition.

Knutsson, H. (1989). Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis*, pages 244–251, Oulu, Finland. Report LiTH–ISY–I–1019, Computer Vision Laboratory, Linköping University, Sweden, 1989.

Kohonen, T. (1989). *Self-organization and Associative Memory.* Springer–Verlag, Berlin, third edition.

Landelius, T. and Knutsson, H. (1993). The Learning Tree, A New Concept in Learning. In *Proceedings of the 2:nd Int. Conf. on Adaptive and Learning Systems*. SPIE.

le Cun, Y., Denker, J. S., and Solla, S. A. (1990). *Advances in Neural Information Processing Systems 2*, chapter Optimal Brain Damage, pages 598–605. Morgan Kaufmann. D. Touretzky, editor.

Meehl, P. E. (1950). On the circularity of the law of effect. *Psychol. Bull.*, 47:52–75.

Mendel, J. M. and McLaren, R. W. (1970). *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, chapter Reinforcement learning control and pattern recognition systems, pages 287–318. Academic Press. J. M. Mendel and K. S. Fu, Eds.

Mikaelian, G. and Held, R. (1964). Two types of adaptation to an optically–rotated visual field. *American Journal of Psychology*, (77):257–263.

Minsky, M. L. (1963). *Computers and Thought*, chapter Steps Towards Artificial Intelligence, pages 406–450. McGraw–Hill. E. A. Feigenbaum and J. Feldman, Eds.

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*, page 108. Prentice Hall Inc.

Mitrani, I. (1982). *Simulation techniques for discrete event systems.* Cambridge University Press.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–293.

Moore, A. W. (1990). *Efficient Memory-Based Learning for Robot Control.* PhD thesis, University of Cambridge, England.

Morgan, J. N. and Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *J. Amer. Stat. Assoc.*, 58:415–434.

Mozer, M. C. and Bachrach, J. (1990). Discovering the structure of a reactive environment by exploration. *Neural Computation*, 2:447–457.

Narendra, K. S. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1:4–27.

Narendra, K. S. and Thathachar, M. A. L. (1974). Learning automata - a survey. *IEEE Trans. on Systems, Man, and Cybernetics*, 4(4):323–334.

Omohundro, S. M. (1987). Efficient algorithms with neural network behavior. *Complex Systems*, 1:273–347.

Omohundro, S. M. (1990). Geometric learning algorithms. *Physica D*, 42:307–321.

Omohundro, S. M. (1991). Bumptrees for efficient function, constraint, and classification learning. Technical report, International Computer Science Institute, Berkeley, California, USA.

Omohundro, S. M. (1992). Best-first model merging for dynamic learning and recognition. Technical report, International Computer Science Institute, Berkeley, California, USA.

Pavlov, I. P. (1955). *Selected Works*. Foreign Languages Publishing House, Moscow.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1986). *Numerical Recipes*. Cambridge University Press.

Reddy, R. (1988). Foundations and grand challenges of artificial intelligence. *AI Magazine*, pages 9–21.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Sirat, J. A. and Nadal, J.-P. (1990). Neural trees: a new tool for classification. *Network*, 1:423–438.

Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis*. Prentice–Hall, Englewood Cliffs, N.J.

Skinner, B. F. (1950). Are theories of learning necessary? *Psychological Review*, 57:193–216.

Sorenson, H. W. and Alspach, D. L. (1971). Recursive bayesian estimation using gaussian sums. *Automatica*, 7:465–479.

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

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the 7th Int. Conf. on Machine Learning*, pages 216–224.

Sutton, R. S. and Werbos, P. J. (1990). *Neural networks for control*, chapter General Principles, pages 38–42. Cambrdige Mass, MIT Press. W. T. Miller, Ed.

Tesauro, G. (1987). Scaling relationships in back-propagation learning: dependence on training set size. *Complex Systems*, 1:367–372.

Tesauro, G. (1990). Neurogammon: a neural network backgammon playing program. In *IJCNN Proceedings III*, pages 33–39.

Thorndike, E. L. (1898). Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review*, 2(8). Monogr. Suppl.

Thorndike, E. L. (1911). *Animal Intelligence.* Macmillan, New York.

Thorndike, E. L. (1932). *The fundamentals of learning.* Teachers College, New York.

Thrun, S. B. and Möller, K. (1992). *Advances in Neural Information Processing Systems 4*, chapter Active exploration in dynamic environments. Morgan Kaufmann, San Mateo CA. J. E. Moody and S. J. Hanson and R. P. Lippmann, Eds.

Tollman, E. C. and Brunswik, E. (1935). The organism and the causal texture of the environment. *Psychological Review*, 32:43–77.

Valtonen, K. (1993). An artificial neural network using pulsed signals. Technical Report LiU-Tek-Lic-1993:1, ISY, Linköping University, S–581 83 Linköping, Sweden.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280.

Walker, S. F. (1984). *Learning theory and behaviour modification.* New Essential Psychology. Methuen, London and New York. Peter Herriot, Eds.

Watson, J. B. (1914). *Behaviour: An Introduction to Comparative Psychology.* Holt, Rinehart & Winston, London.

Whitehead, S. D. (1991). A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, Computer Science Department, University of Rochester.

Whitehead, S. D. and Ballard, D. H. (1990). Active perception and reinforcement learning. *Proceedings of the 7th Int. Conf. on Machine Learning*, pages 179–188.

Widrow, B. (1987). Adaline and madaline - 1963. In *IEEE First Int. Conf. on Neural Networks*, pages 143–157.

Williams, R. J. (1987). A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *IEEE First Int. Conf. on Neural Networks*, pages 601–608.

Wilson, R. and Knutsson, H. (1993). Seeing Things. Report LiTH-ISY-R-1468, Computer Vision Laboratory, S–581 83 Linköping, Sweden.

Wittgenstein, L. (1958). *Philosophical Investigations.* Basil Blackwell, London.