Applying Neural Networks

Jussi Stader

AIAI-IR-11

August 1992

Abstract

The claims made about neural networks' strengths and weaknesses vary. This report intends to put these claims into perspective. There is a wealth of connectionist methods and techniques each suited to different tasks, each having different problems. Some of them are well understood, others are not. It is important to identify and assess both the weaknesses and the strengths of neural networks in order to determine what they are capable of. Given the capabilities of neural networks application areas can be identified. In this report special attention is paid to the application of neural networks in relation to traditional Artificial Intelligence.

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Contents

1 Introduction

Neural networks are systems composed of a large number of highly interconnected processing units. They are inspired by the physiology of biological nervous systems. Neural networks are often classified as Artificial Intelligence (AI). In this report they are treated as a separate field from the traditional symbol processing AI of knowledge based systems, robotics, natural language processing and related areas.

For the past fifty years neural network research has been carried out at various Universities and Institutes. Recently the interest in neural networks has increased. This sudden exposure has led to high expectations of what neural networks can do for different fields, especially fields where other approaches have not been applied successfully. However, it is often difficult to distinguish between results which provide sound technology and unfounded claims or premature exploitation of findings that need to be researched further.

The most common view of a neural network is that of a system that does not have to be programmed – it only needs to be taught a problem area and it will then miraculously solve any similar problem. Often this leads to systems that nobody understands: nobody knows what exactly the system does, what problem the system solves, how the system works and what exactly it can be used for. This is a serious problem, especially because it is not clear what kind of problems can be solved by using neural networks and which cannot.

This report intends to clarify some of the claims that have been made about neural networks and outline those methods and application areas which are understood well enough to provide a good chance of success. It is organised in seven sections. Following this introduction, section 2 describes the main learning strategies and network architectures and discusses their theoretical foundations, limitations and suitable application areas. The reader who is already familiar with the different neural networks architectures and techniques may want to skip most of this section. However, some of the applications, theory and problems details in section 2.2 and 2.3 may still be of interest. Section 3 outlines the main problems of neural networks, including development and design problems as well as operational problems. In section 4 the strengths of neural networks are described. Section 5 and 6 show what applications neural networks are suitable for. Section 5 gives a general idea, while section 6 concentrates on applications connected to traditional AI. Finally, section 7 presents conclusions.

2 Connectionist Methods and Techniques

This section describes the basic building blocks of connectionism [1] to give the reader some background on the technology for further evaluation and discussion. There is a wealth of network architectures and learning algorithms that can be used, each with their own advantages and disadvantages and each suited best to solve different problem classes.

In the following there is a brief description of the components and operation of neural networks. Then, in section 2.2, different methods for teaching neural networks are described: supervised and unsupervised learning, and back-propagation and its derivatives. Finally, in section 2.3, some of the major network types are discussed: Perceptrons, multi-layer Perceptrons, Hopfield nets, Boltzmann machines, associative nets, adaptive logic networks, Kohonen's self organising feature maps and cascade correlation.

A more detailed description of the different methods and techniques can be found in [RM86, Byt87, AR88, Hin89, Was89, Hin89].

2.1 Basics

This sub-section describes the structure of neural networks, their operation and the representation of information within neural networks.

2.1.1 Structure

Units: The basic building blocks of neural networks are units (sometimes called nodes or neurodes). They are inspired by the neurons found in the nervous system although they are simplified. A unit is a simple processor that uses a function to compute its state of activation and its output from an array of input values it receives. Units can be binary (0,1 or -1,1) or analogue (continuous value range).

Networks: Neural networks are made up of units that are interconnected (analogous to neurons with synapses). Connections between units have weights which can be modified. Weights determine how strong a connection between two units is. Connections can be excitatory (supporting activation) or inhibitory (suppressing activation).

Units may be organised into layers (structured networks). A net can have one input layer, one output layer and any number of hidden layers. Hidden units are units that

[1] Note that in this report no distinction is made between the terms 'connectionism' and 'neural networks'. In some of the literature 'neural networks' refers to more physiological systems whereas 'connectionism' is used for computationally inspired systems. 'neural networks' are also refferred to as 'artificial neural networks', or ANNs.

are not used for input or output of the network. They can be seen as holding the network's own interpretation of the dependencies between input and output patterns. In the literature the input layer is sometimes not counted as a layer because the input units have no functionality. Their states are simply set to the corresponding value of the input vector (pattern).

There are two basic types of network: feed forward networks and feedback networks. In a feed forward network there are no cycles, i.e. no unit receives input from any unit that its output is connected to (directly or indirectly). In feedback networks cycles are allowed. This makes it possible for a network to have some kind of memory.

The overall behaviour of the network depends on its structure and on the strength of connections (weights on connections).

2.1.2 Operation

Neural networks are developed as follows: first the network is set up with all its units and connections and it is usually initialised with arbitrary weights. Then the network is trained by presenting examples. During the training phase the weights on connections are changed which enables the network to learn. There are different learning algorithms that can be used to modify weights. When the net performs well on all training examples it should be tested on examples that it has not seen before. If the net can deal with these test cases too, it is ready to be used. Some networks then stop learning and are used for problem solving only, whereas other networks continue to update their weights whenever a problem is presented to them.

On a more detailed level neural networks operate by propagating activation states of units across the net. This is done by letting each unit re-compute its output. Units compute their output using the weights on the connections from their (pre-synaptic) input units, and the activation states of those input units. Some units have a threshold which is used to decide whether the unit has received enough excitatory input to be activated.

The propagation of activation can be done using different scheduling algorithms. These algorithms determine which unit re-computes its activation at which time, and when the new activations take effect. Examples for scheduling algorithms are:

- fixed order: each unit is updated in turn. Updates take effect immediately.
- random order: picks a unit to be updated at random, making sure that each unit is updated at some point. Again, updates take effect immediately.
- simultaneous: a snapshot of the network's activation is used for the computation. Note that the new outputs are ignored until all units have been updated.

An epoch is a presentation of the whole set of training examples to the net. Training times are usually measured in epochs. There are two major strategies for updating weights during training:

1. to collect the error information and update the weights after an epoch,
2. to update the weights after each training pattern is presented.

2.1.3 Representation

In neural networks information is represented in weights and the network's state, i.e. the states of the network's units. The representation can be local or distributed. In local representations each unit has a well defined task in the representation. The tasks of different units do not overlap to any great extent. The resulting cells are also known as "grandmother cells", indicating that there would be one cell that represents the concept of a grandmother. In distributed representation many units are used to collectively represent a concept, and each unit is used to represent part of many concepts.

One advantage of distributed representation is that it is more robust towards damage. If a network with local representation loses a unit, the concept which is represented by that unit will no longer be available. In distributed representation the loss of a unit removes only a small part of a concept, not the concept itself. One way around this problem is to use non-overlapping unit pools or clusters for local representation instead of single units. Each unit pool is then a dedicated local representation of the concept. Distributed representation generally uses less units than local representation. It can be compared to the ASCII encoding of the character set which uses eight bits. With local representation there would be as many bits as there are characters. A disadvantage is that there may be 'cross talk' in distributed representations. If the representation of two different patterns shares too many units, the patterns cannot be distinguished reliably.

Note also that the granularity of the language to be represented is important. A concept in a high level language can be divided into many concepts in a low level language: a high level word corresponds to a low level sentence.

The representation in neural networks is very important. What is considered to be the "best match" depends on the domain. How easy it is to find this best match depends on the way things are represented. For example one animal instance can mean food for one other animal, companion for the next and predator for the third. The representation must be such that similar things in the domain sense look similar in the representation. This is not only true for pattern matching but also for problems like generalisation and classification.

2.2 Teaching the Nets

Probably the most intriguing feature of neural networks is their ability to learn. There is a variety of different learning methods that can be used to train neural networks. There are two major learning techniques: supervised learning where a teacher provides inputs and desired outputs for training examples, and unsupervised learning where the network learns to solve a problem by only looking at input examples without the guidance of a teacher.

Most neural networks use some form of *Hebbian learning* to modify the strengths of their connections through weights. Hebb's postulate [Heb49] is

> When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

This means that connections between units become stronger if the units' activities are correlated. Unsupervised Hebbian learning can produce receptive field properties as found in the visual cortex.

An important issue for learning in neural networks is the *credit assignment problem*. This is the problem of determining whether and how a unit contributed towards an output error (or an output success). In simple network architectures it is possible to assess each unit's contribution locally. The only information needed is the unit's own activation and the activation of its input units. The weights on the incoming connections can then be modified accordingly. More complex networks that have hidden layers are more powerful, but they are more difficult to train. The error of the output units must be propagated through to the hidden layers in order to be able to assess the performance of the hidden units. Backpropagation is one example of how errors can be propagated through a network with hidden units. It is described in section 2.2.3. Backpropagation can be used for (self)supervised learning in networks with hidden units.

2.2.1 Supervised Learning

In supervised learning an evaluation of the output which the network produces is available for the training examples. Supervised learning methods differ in how much information is available about the output. Most commonly a teacher specifies both the input and the desired output of the learning network. Weights are changed to reduce the error which is the discrepancy between the desired output and the output the network produces itself. Usually weights are modified using some form of Hebbian learning.

In multi-layer networks it is necessary to propagate the error backwards through the net so that the contribution of hidden units towards the overall error is taken into account. This credit assignment problem can be solved using techniques like backpropagation. If the desired states of all hidden units are known for the training examples, then the credit assignment problem can be solved locally and no backpropagation of errors across different layers is necessary. This is called *easy learning* as opposed to *hard learning* where only the final output of the network is given.

Self-supervised learning is a special case of supervised learning where the desired output is the same as the given input. Therefore no teacher is needed for supervision.

In *reinforcement learning* the desired output of the network is not known, but a scalar evaluation of the network's calculated output is available. The evaluation provides a measure of how good the result is which the network produced. Credit is assigned to local decisions of units by measuring the decision's relation to the global reinforcement signal.

Applications: Supervised learning should be used in cases where there is sufficient knowledge about the data that the network will encounter. It must be known what response the network should give for each input pattern, which means that a solution to the problem must already be known. Generally supervised learning is used for learning tasks that people can perform, but non-human teachers are also possible (e.g. observe a system in operation when learning to simulate it).

Networks trained with self-supervised learning can be used as associative storage: they can complete partial patterns, eliminate noise from data, correct errors, reduce dimensionality (data compression, reversible encoding), generalise, etc.

Problems: The major problem of supervised learning is that the correct solution is required during learning. This desired output cannot be provided for all problems.

2.2.2 Unsupervised Learning

Unsupervised learning is usually competitive, based on the winner-takes-all strategy: the unit with the biggest total input wins and is activated, all other competing units are de-activated. The winning unit updates its incoming weights so that connections from active units are made stronger and connections from inactive units are made weaker, so that the weight vector is made more similar to the input vector. This way the winning unit is made more sensitive to that input vector and the next time that unit will win more convincingly. Note that only the winning unit learns, the rest of the network remains unchanged.

Applications: Unsupervised competitive learning divides a set of input patterns into disjunct clusters. Each cluster contains patterns with similar features. Therefore these networks can be used to detect regularities in input patterns. They can determine which features are relevant for classification, reduce dimensionality and enhance contrasts.

Output units can be clustered themselves, each cluster forming a small winner-takes-all net so that a more distributed output is achieved. These nets can be scaled up more easily to solve more difficult problems.

This learning method is particularly suitable where input is taken directly from the environment and the desired output cannot be provided for training.

Often in networks trained with unsupervised learning the distinction between training and operation is less pronounced: the networks continue to learn when they are being used to perform their particular tasks. This makes them suitable for autonomous systems that have to adapt to a changing environment without being re-programmed.

2.2.3 Back-Propagation

Description: The backpropagation algorithm is a generalisation of the least mean square algorithm (LMS). It uses gradient decent to minimise the cost function of the net which is the mean square difference between the desired output and the actual output.

For output units the error depends on the difference between the actual and the desired output. For hidden units the error is computed using the unit's own activation and the weighted error of all its (post-synaptic) output units in the layer above. Thus the error is propagated back across the layers of the net and weights are changed accordingly.

Backpropagation can be used in feed-forward networks for supervised or self-supervised learning.

Operation: The network is initialised with small random weights and the input is presented. When the network has produced its output this is compared to the desired output. Starting with the output units the error of each unit is computed and the weights of the unit's incoming connections are changed according to the error.

Applications: backpropagation has been used for multi-layer networks in many applications. It has been used for deterministic problems like the logical operator exclusive-OR (XOR), which is true only if exactly one of its inputs is true. Backpropagation has also been used in speech synthesis and recognition, visual pattern recognition and many other applications.

Theory: Back-propagation is well understood for simple cases, but it is not clear where the boundaries are to what kinds of problems it can solve. Back-propagation is mainly characterised by observations of its applications, which is not a strong theoretical basis. However, back-propagation performs well on small problems and it finds good solutions.

Problems: The biggest problem of back-propagation is that it is slow on large networks. This seriously limits the applications it can be used for. The learning time increases with the number of connections (i.e. weights to be updated) and number of hidden layers. Learning becomes less effective with more hidden layers because the error signal generated by the output is diluted. A good way to affect learning time is to change the step size. For fast learning the steps towards the error minimum should be big. However, with big steps the minimum may be missed. Back-propagation does not observe how the error surface changes and therefore large steps are risky.

During learning the only information a unit receives is its input and its error signal which is propagated back from the network's output. The unit's input is the data, the error signal is the description of the unit's task. The unit tries to change its incoming weights in order to learn its task. During training with back-propagation a unit's error signal changes frequently which means that its task specification changes. This is known as the *moving target problem*: the problem a unit is trying to solve changes all the time. This is due to the fact that other units update their weights at the same time.

Back-propagation uses gradient descent to reach a solution. This makes it impossible to produce cognitive behaviour that requires a temporary disadvantage in order to achieve an overall success. For example consider the problem of reaching food on the far side of a fence. If there is a hole in the fence some distance away, then the solution can be reached by initially increasing the distance to the food, which is an immediate disadvantage. Not being able to produce this behaviour is a limitation on what problems can be solved using back-propagation (or any other gradient descent method).

A somewhat disturbing problem is that output units tend to get stuck in an inactive state during training with back-propagation. This problem is enhanced when more hidden units are used.

Finally backpropagation is biologically implausible. It corresponds to using synapses in reverse which is a highly unlikely theory for learning in biological neurons.

2.2.4 Derivations of Back-Propagation

Back-propagation is slow on large and complicated tasks. It scales up badly for networks containing more units with more connections or more layers. Since slowness gets worse with larger nets this limits the scope of the applications to small scale problems.

One approach to overcoming this problem of network size is to eliminate connections between un-correlated units. This way there are fewer weights to be changed during back-propagation which gives more efficiency. However, this approach does not change back-propagation itself, it only pushes the limitations a little further away.

Back-propagation performs gradient descent in weight space to reach a minimum of error. An important parameter is the step size used for the descent. Speedups are possible through taking steps as big as possible downhill but making sure not to over-shoot the minimum.

Learning with gradient descent can be improved by adjusting the learning rate dynamically based on the history of the computation and some heuristics. A momentum parameter can be used for this adjustment, increasing it if two successive gradient vectors are very much alike, decreasing it otherwise. Adjustments can be made globally for the whole net or locally for each weight.

Here are two examples of improvements of back-propagation:

Quickprop: [Fah88] Quickprop is a derivative of back-propagation that improves learning speed. Back-propagation is done as normal, but for each weight the error derivative of the previous epoch is kept in addition to the current error and the last change of the weight value. Two pragmatic assumptions are made: 1. the error function is a parabola, so there is no chance of over-shooting into an adjacent valley and 2. other weights have no influence on the local error function. This approach gives an approximation for the best weight update. Basically quickprop looks at the slope in the weight error. If it has the same direction as for the last weight change this means that the system is still moving towards the error minimum. The opposite direction means that the minimum has been passed. Quickprop performs more than a magnitude better on data compression problems than back-propagation.

Backpercolation: Backpercolation is another derivation of back-propagation which adds a local error assignment for each unit to back-propagation's error gradient. This way each unit has information about direction *and* distance for the next weight update, rather than just direction. Convergence speed is increased and local minima are avoided more successfully.

2.3 Net Types and Architectures

In this subsection different network types are described in some detail. The descriptions are organised as follows:

Description: type of units, net structure (connectivity), weights, input values and output values,
Operation: behaviour of the net including general description of training and retrieval phases: initialisation, input presentation, learning (weights update), output calculation, learning method and adaptation,
Applications: characteristics, areas of usefulness,
Theory: theoretical foundations: analogies, proofs,
Problems: limitations, shortcomings, difficulties.

2.3.1 Perceptrons

Description: Perceptrons are the simplest network architecture. There are no hidden units and only one output unit which is connected to all input units. Connections are excitatory and have modifiable weights.

Input units have continuous state values, whereas the state of the output unit is binary (1,0 or 1,-1). The output unit usually has a hard delimiter nonlinearity and a threshold. The input units are used solely to present the input pattern to the net.

Operation: Initialised with arbitrary weights Perceptrons are trained by supervised learning: an input pattern is presented to the net, the output is calculated and compared to the desired output. The weights are changed according to the output error. Then the next training example can be learned. This procedure is repeated until the weights converge and do not need to be changed any more. After that the net can be used on unknown input patterns for which it will generate the appropriate output.

The output is calculated by computing the weighted sum of the inputs, subtracting the unit's threshold and applying the hard delimiter nonlinearity.

The weights are updated using a very simple learning algorithm: if the calculated output is correct, no weights are changed. Otherwise the weights of connections to active input units are changed to support the desired response: if the output unit should be active the weights are increased, if it should be inactive the weights are decreased.

Applications: Perceptrons can be used as pattern classifiers for simple problems using the two possible output values to indicate two different classes.

In order to understand more precisely which kinds of problems Perceptrons can tackle, it is helpful to look at the space of inputs which is spanned by the input variables (bits of the input vector). Each input pattern can be described as a point in this space. In a classification problem those points in the input space belonging to one class are to be separated from those belonging to the other class. The Perceptron approach to separation is to place a hyperplane in the input space. This means that they can solve problems that are linearly separable. Examples of linearly separable problems are performing the logical AND and OR.

Theory: Perceptrons are simple networks and they are well understood. It has been proved [AR88] that if it is possible to separate the two classes in the input space by a hyperplane then Perceptrons converge.

Perceptrons can be used to implement the Least Mean Square (LMS) algorithm by replacing the hard delimiter nonlinearity of the output unit by a threshold logic nonlinearity.

Perceptrons are also similar to the maximum likelihood Gaussian classifiers, which have similar decision regions. However, the Gaussian classifier makes assumptions about the underlying distribution of the input.

Problems: Only a small set of problems can be solved by placing a hyperplane in the input space. An example of a problem that cannot be solved this way is the logic XOR operator. An attempt to solve such a problem may cause the Perceptron to oscillate instead of converging.

Thus Perceptrons are not powerful enough for most real world problems.

2.3.2 Multi-Layer Perceptrons

The limitations of the original single layer Perceptrons have lead to the development of multi-layer Perceptrons.

Description: multi-layer Perceptrons are feed-forward nets like Perceptrons but with one or more hidden layers. To give the net a wider range of application, multiple output units are used together with sigmoidal nonlinearities. The output units can now have continuous states which leads to smooth decision regions. Learning in multi-layer Perceptrons can be supervised or self-supervised.

Operation: Multi-layer Perceptrons are more powerful than Perceptrons, but they are also more difficult to train. Multi-layer Perceptrons are trained using supervised or self-supervised learning with the help of backpropagation. During training every trial consists of two phases: a forward sweep for propagation of activation and a backward sweep for error propagation by changing weights. The training patterns (together with the desired outputs) are presented repeatedly until the weights converge and the error of the net is reduced to an acceptable level.

Applications: In contrast to the original Perceptrons which have half plane decision regions, Perceptrons with one hidden layer have convex decision regions which are formed by the intersection of the decision regions of the hidden units. At least one hidden unit is needed for each side in the decision region. However, if the decision regions are disconnected or overlapping, a second hidden layer is needed. With such a three layered net the decision regions can be separated arbitrarily.

The decision regions of multi-layer Perceptrons suggest what kinds of problems can be solved this way. The disconnected decision regions of three-layer Perceptrons show that these nets can be used for mappings with regularities and exceptions.

Multi-Layer Perceptrons have been used with success on

- deterministic problems, e.g. XOR,
- speech recognition,
- speech synthesis,
- visual pattern recognition,
- control of autonomous vehicles,

and many more. See also section 5.

Theory: The fact that multi-layer Perceptrons are more powerful than single layer Perceptrons is due to the nonlinearities of the units. If these were abandoned, any net with hidden layers could be reduced to a net without hidden layers.

The tradeoff for making Perceptrons more powerful is that the analysis of the net becomes more difficult. The characterisation of multi-layer Perceptrons is mainly done by observation. It has not been proved for which cases these nets do not converge, which is one of the problems with this approach. The nets can solve some problems well but it is not clear where their limitations are.

It has been proved that a three-layer Perceptron with N(2N+1) units that have continuously increasing nonlinearities can compute any N-ary function. However, this only ensures that such a net exists; it does not describe the net and little is known about how to build such a net.

Problems: As mentioned above one of the problems of multi-layer Perceptrons is that it cannot be proved when they converge. There are rules about how many units are needed in hidden layers for a particular problem given the problem's decision region. In general the number of hidden units influences the speed of convergence, the generality of the net and its fault tolerance. If the net has many hidden units it converges quickly, but the hidden units will contain a representation tuned to the training patterns (the network has memorised the examples without generalising). If the net has few hidden units it will converge slowly, needing a lot of training cycles, but the hidden units will provide a good generalisation. If there is no redundancy in the net it becomes less fault tolerant because all units and connections are needed to solve the problem. Note that the net will not converge if there are too few hidden units.

Pragmatically a more serious problem is that these nets are slow on learning tasks of a reasonable size because of the number of weights to be updated and because of the computations necessary for these updates.

2.3.3 Hopfield Nets

Description: Hopfield nets are unstructured nets. Their units are not organised into layers. The connections between units are symmetric: if there is a connection from unit a to unit b then there is also a connection from unit b to unit a and the two connections have the same weights. Connections between units can be excitatory or inhibitory. Units usually have discrete states 1 and 0 (or -1).

Operation: Hopfield nets are trained by self-supervised learning. The weights are set so that each example pattern is represented by a stable state. This can be done by calculating the weights on all connections using the set of examples to be learned. Given a new input pattern the net will then settle into the (previously learned) stable state closest to the given input pattern. An input pattern is presented to the net by setting the states of the net's units (active/inactive). Note that in a Hopfield net the two stages of training and retrieval are distinct: Hopfield nets are not adaptive. The learning phase is always terminated before the retrieval phase begins.

The net changes its overall state by asynchronous iteration, updating the states of its units by computing their activation function. Usually the activation function is simply the weighted sum of the incoming connections. With each iteration the states of units are changed, which influences other units' states at the next iteration. Through these iterations the net converges, i.e. it reaches an overall stable state where the states of units stop changing. When a Hopfield net is in a stable state, its energy measure is minimised. The energy measure is the potential for change in unit's states within the net.

During training the weights are assigned to connections according to the elements of the training samples. If two units are often active simultaneously the connections between them are strongly excitatory; if mostly one unit is active while the other is not then the connections are strongly inhibitory; and if two units are mostly independent the connections are weak.

Applications: Starting from different input patterns the net can reach different stable states depending on the training samples. In general a Hopfield net reproduces that training sample that is closest to the current input pattern. This way the net can be used as associative storage for completion of partial patterns, recognition of noisy patterns or correction of faulty patterns.

If the training samples are typical cases of different classes the net can be used for classification and generalisation.

Another use for Hopfield nets is diagnosis. The units in such a net represent different characteristics (or symptoms) and hypotheses and the weighted connections between units represent compatibility between hypotheses or between characteristics and hypotheses. Given a set of characteristics, the net settles into a state where the set of compatible hypotheses is active.

Finally Hopfield nets can be used for constraint satisfaction and for optimisation problems like the traveling salesman problem where the shortest path is to be found that will visit a given set of locations. For constraint satisfaction a minimum of energy corresponds to a maximum of satisfied constraints. Applications include circuit layout and load balancing.

Hopfield nets perform best if not too many patterns are stored and those patterns stored are sufficiently different from each other.

Theory: Hopfield nets are quite well understood. Some theorems have been proved about nets reaching a stable state and the computation capacity for energy minimisation has been examined. The fact that Hopfield nets bear a similarity to stochastic multi-variable optimisation and to spin glasses helps the theoretical understanding of this technique.

Their use as associative storage has been well investigated. Their storage capacity depends on how dissimilar the patterns to be stored are. If the patterns to be stored are orthogonal (as dissimilar as possible), then the dimensionality (number of input bits) of the patterns is the upper limit to the capacity. If patterns are not orthogonal then the storage capacity is between 10% and 15% of the dimensionality.

Problems: Hopfield nets are an implausible model of real neurons and the brain because of their symmetric connections. A more serious problem is that Hopfield nets can get trapped in local minima of their energy measure thus giving a sub-optimal result. A possible improvement is to use units with analogous states.

When used for associative storage, the limited storage capacity is a problem, as is the fact that the net's performance depends on the type of patterns stored: performance is better if patterns are dissimilar.

2.3.4 Boltzmann Machines

Description: Boltzmann machines are generalisations of Hopfield nets. They are adaptive nets that can modify their connectivity and they have hidden units which are neither input nor output units. Units have thresholds and they are often analogous, i.e. their output varies continuously (sigmoid) with applied voltage. As in Hopfield nets, connections are symmetric and can be excitatory or inhibitory.

An important concept for Boltzmann machines is the energy of the net, which is a measure of the conflicts between connected units. A conflict exists if connected units have incompatible states so that a unit's activation is inconsistent with its input.

Operation: Initialised with arbitrary weights, Boltzmann machines are trained with supervised learning: first input and output patterns are presented to the net, then the output is removed and the weights are updated according to the difference in the net's state. The units' states are updated until equilibrium is reached (the units stop changing state and the net is in an overall stable state). Input and output patterns are presented to the net by clamping the units to the desired states.

Units have probabilistic activation functions. A unit's state is determined as follows: first the weighted sum of the inputs is calculated. This is then compared to the threshold to give the energy gap which is the difference between the unit's activation and the threshold. Then temperature is introduced. Together with the energy gap the temperature determines the probability of a unit being active in the new state. The temperature introduces a random aspect to the behaviour of units' states. With a high temperature the probability of a unit being active depends only to a small extent on the unit's input. Thus almost random behaviour of the net can be achieved.

During learning the probability for units being active is used to update weights. With the input and output units clamped to the desired states the states of hidden units are updated until the net reaches equilibrium. For each connection the probability of both connected units being active is calculated. Then the output is removed. This time not only the hidden units are updated, but also the output units. Again for each connection the units' activation probabilities are calculated. Weights are changed according to the difference between the probabilities calculated with the given output and those calculated with the net's own output.

During operation Boltzmann machines work by minimising the net's overall energy. This is done by iteration, asynchronously updating units' states. In addition Boltzmann machines use the temperature to avoid local minima. With a high temperature the net moves fast towards its equilibrium, but high and low energy states have roughly the same probability. Thus the net behaves almost randomly. At low temperatures the net moves slowly towards its equilibrium. Low energy states are more probable than high ones. Eventually the net behaves almost deterministically, much like a Hopfield net. A successful strategy for using temperature to avoid local minima is simulated annealing: starting with a high temperature to allow almost random jumps, the temperature is decreased slowly making the net behave more and more like a gradient descent system to ensure that eventually the system settles in a minimum.

Applications: Boltzmann machines can be used for the same problem types as Hopfield nets. However in cases where global minima are preferable to local minima Boltzmann machines are superior. Additionally the fact that Boltzmann machines can have separate units for input and output enables them to associate different patterns whereas Hopfield nets are restricted to pattern recognition, reproducing a pattern that was stored. Hidden units that are not used for input or output allow Boltzmann machines to represent complex correlations between input and output patterns.

Boltzmann machines can also simulate multi-layer Perceptrons and therefore they can be used as feature detectors, encoders, etc. (see above).

One advantage of Boltzmann machines (and Hopfield nets) is that they are easy to implement in hardware.

Theory: Boltzmann machines are fairly well understood. Some of the techniques that are used have been analyzed theoretically, others statistically. For example it is relatively easy to show that the way units are updated causes the net's energy to decrease monotonically [2]

Problems: As with Hopfield nets Boltzmann machines are not a plausible model of neurophysiology, but the major problem of Boltzmann machines is that they are slow, both during learning and during operation. A more practical problem is that hidden units tend towards being active, which can cause serious problems in development.

[2]The energy of the new state is less than or the same as the current energy

2.3.5 Willshaw Nets

Willshaw nets are nonlinear associative nets. They have binary inputs, binary weights and outputs with thresholds. There are no hidden units.

Willshaw nets can best be visualized as a connection matrix with horizontal input lines (axons) and vertical output lines (dendrites). The matrix contains binary values (0 or 1) to indicate whether a connection exists between an input unit and an output unit. These values are the binary weights.

Operation: Initially all connections in the net are off (the connection matrix contains only 0). Willshaw nets are trained by supervised learning: both input and output patterns are presented to the net. According to these patterns the connections are updated: a connection is switched on (a synapse is generated) if both the input line and the output line are active in the training patterns. Once a connection is switched on it is never switched off again. Each training example needs to be presented to the net only once.

When the net has learned all its training examples, the input-output associations are stored and the net can then be used. If an input pattern is presented to the net it will generate the associated output pattern. This is done by determining the activity of each output line by summing its active on-connections (connections to active input lines). If this sum is greater than the line's threshold the output line is active.

The representation of stored patterns is distributed in the connections. If too many patterns are stored, i.e. the net's storage capacity is exceeded, the network's performance degrades and errors occur during association: the wrong response is generated for a given input.

Applications: Willshaw nets are used as pattern associators in two basic ways: for auto-association using self-supervised learning and for hetero-association where the net learns to associate two different patterns. This process is also known as conditioning.

An advantage of Willshaw nets is that connections are minimised. Only pairs that influence each other are connected.

Willshaw nets are easy to implement because input units and weights are binary and only the output units are graded.

Theory: Willshaw nets are well understood. They are a derivative of holographic memories. The information storage capacity of Willshaw nets has been determined: it is proportional to the number of possible connections.

Problems: If too many patterns are stored the association degrades

2.3.6 Kohonen's self organising feature maps

[Koh82, Koh88] In the brain there are many cases in which aspects of sensory input are represented in two dimensional areas of the cortex. For example, the visual space is mapped onto the cortex surface retaining relative positions. There are several such topographic mappings, each specialising in different sensory aspects like shape, colour and movement. These topographic maps within the brain are the inspiration for Kohonen's topographic feature maps. The importance of global organisation is recognised on top of the functionality of individual units.

Description: Kohonen's self organising feature maps are associative networks. The nets are organised in a two-dimensional array. Units are connected so that there are excitatory connections between nearby units and inhibitory connections between distant units.

Operation: Initially all units respond randomly to the input parameter (e.g. frequency, colour, ...). The network is trained with an adapted form of the winner-takes-all strategy of unsupervised learning. An input pattern is presented to the net and the unit with the highest response to that pattern is located and its weights are updated. Additionally the unit's neighbours are identified, i.e. those units that are located in some region around the winning unit. These neighbouring units also have their weights updated. This way the formation of neighbourhoods is encouraged so that nearby units respond similarly to inputs.

This architecture and learning process results in fast and reliable organisation of networks. Usually the system will start organising into large regions and focusing into smaller regions during training.

Applications: Kohonen's feature maps are modeled after properties of the cortex. In the brain topographic mappings are used in the visual system, for touch sensors in the body surface, in the auditory system, and for similar sensory representations.

Natural applications for Kohonen's feature maps are those where global organisation of parameters is relevant, such as object and character recognition, feature detection based classification and contour completion. Feature maps have been used for speech recognition by representing acoustic cues of phonemes across the network surface. Words are recognised using their paths across this phoneme space.

Theory: In simple cases it can be proved that feature maps will organise properly. Probably the best argument for these systems is that they are used in the brain for representation of the sensory environment and therefore they must be useful for representing this type of information in connectionist systems. The connections used in Kohonen's feature maps can also be found in teh brain: often neurons excite nearby cells and inhibit far away ones. Finally, the learning method used is a simple adaptation of Hebbian learning which is an accepted model of learning in nervous systems. Thus biological plausibility includes the architecture, the learning mechanisms and the functionality of the resulting system.

Problems: Little is known about the properties of large systems. The similarity to aspects of organisation within the brain cannot be used as a strong theoretical foundation. It is not clear how large and complicated a task can be solved this way.

2.3.7 Cascade Correlation

The problems of back-propagation have led to the development of cascade correlation networks. These networks address the problems of network architecture, learning speed and task complexity [FL90].

Description: Cascade correlation networks are feed forward networks. They have input and output units and they may have hidden units. Output units and hidden units have sigmoid activation functions taking states between -1 and 1. Cascade correlation networks are fully connected, i.e. each input unit is connected to each output unit. If there are hidden units they receive inputs from all input units and all hidden units in previous layers. Their outputs are fed into all output units and all hidden units in successive layers. Each hidden layer contains only one unit. Connections have weights that can be modified.

Operation: Cascade correlation networks change their structure during learning. Hidden units are added as needed. For simple problems the network will be able to learn its task without hidden units. For more complex problems hidden units are needed. Initially the network consists of only input units and output units. This rudimentary network is trained by changing weights to solve the task as best as possible, i.e. until there is no significant improvement. The network is then tested to see whether the task has been learned well enough. If there is an unacceptable remaining (rest) error, a hidden unit is added to the network in order to reduce this rest error. The new unit is connected to all input units (and previous hidden units) and all output units. The unit is then trained so that its correlation with the rest error is maximised. Note that this training affects only the new unit's own connections and therefore there is no need for the backpropagation of errors. When the unit has learned to reduce the rest error as much as it can, its input connections are frozen and the network is tested to see whether the task has been learned well enough. If the remaining error is still too large another hidden unit is added, and so on.

A variation of this approach is to train a 'unit pool', i.e. a set of units, on the rest error rather than a single unit. When the units are trained, only the unit that reduces the error most is used in the network, the other units are discarded. Units of different types can be used in one pool, e.g. some units with sigmoid activation functions and others with Gaussian activation functions. The unit type best suited to the current (sub-)problem will win this competition. This way the network can be made up of different special purpose units, each unit contributing to that part of the problem it is best suited to.

The overall strategy of training cascade correlation networks is to start with the connections to the environment (input and output). On top of that a small sub-problem solver is built and incorporated. These sub-problem solvers are built until the network can solve the overall problem. Note however that the overall problem is broken up into sub-problems by the network itself so that the sub-problems are usually not ones an implementor would choose or even recognise.

The main advantages of this approach are: automatic network construction, fast learning, learning of complex behaviour using simple learning methods.

<u>Applications:</u> Weights on connections are frozen when a part of the task has been learned. The learned task is then indifferent to subsequent changes in the network which makes it impossible for a network to 'forget' what it has learned. Thus cascade correlation networks are good for cases where overwriting information is undesirable. Together with their flexible structure this makes cascade correlation networks suitable for incremental learning.

Cascade correlation networks are well suited to problems where it is difficult to decide on the network structure most suitable for learning and performing the desired task. The network will produce its own structure.

Due to their efficiency cascade correlation networks can provide solutions to problems for which other multi-layer networks were infeasible because they were too large or too slow to train. Their modular approach to learning makes them more suitable for complex tasks than other network types.

The general development strategy can be adapted to develop complex feature detectors. During learning the net would successively build useful feature detectors until all required features can be detected.

<u>Problems:</u> The techniques and structures used are less uniform than those used for other network types. These irregularities make it more difficult to analyze the networks. There is also less redundancy in cascade correlation networks because there are only enough hidden units to perform the task, no more. Redundancy in networks makes them tolerant towards damage and noise. Therefore there is less fault tolerance in cascade correlation networks than in less compact architectures.

<u>Recurrent Cascade Correlation:</u> [Fah91] A derivation of cascade correlation adds sequential processing to networks. The feed-forward networks of cascade correlation are turned into feed-back networks by adding each unit's output to its inputs. This way, some memory of previous states is introduced to the network which allows it to learn tasks for which sequence is important (e.g. translate Morse code into letters, learn complicated finite-state grammars).

3 Problems with Neural Networks

There has been substantial criticism of neural networks and periodically there are arguments amongst researchers about whether or not it is worthwhile to study neural nets at all, and whether neural nets offer anything which other techniques cannot provide.

In this section the main problems of neural network techniques are discussed. For this three problem areas are identified:

1. problems of theoretical assessment:
   provability;
2. difficulties in designing neural network systems:
   representation, structure, teaching;
3. problems of working with neural networks:
   interpretation, performance assessment, scale.

3.1 Provability

One of the most serious problems is that for many connectionist techniques there is no firm theoretical basis on which to decide what problems they can solve. In a lot of cases the techniques will work and give good results. However, the confidence in neural networks techniques is reduced by the lack of theoretical proof and the uncertainty of where exactly the line lies between tasks that can be solved and those that cannot. Any paradigm benefits greatly from knowing its limitations. The credibility of systems developed under some technique is enhanced if it is known what problems cannot be solved with those techniques.

As Minsky and Papert continue to point out in [MP88] that the questions to be answered are:

- what can a neural network learn?
- what can it not learn?
- how long does it take to learn it?
- does it get easier or harder with more processing elements?

There are some things that have been proved about connectionist techniques including the following:

- Neural networks are equivalent to Turing machines [MP43, SS91];

- connectionist energy minimisation is equivalent to the problem of satisfiability [Pin90];
- Perceptrons cannot solve the XOR problem [MP88];
- Perceptrons converge for linearly separable decision regions;
- a three-layer Perceptron with the right number and type of units can compute any N-ary function [HSW89];
- theorems about reaching stable states and the computation capacity of Hopfield nets;
- in simple cases Kohonen's feature maps will organise properly;
- capacity of neural networks for associative storage;

see also [AR88, HS87]. In some cases equivalence or similarity of neural network techniques to traditional techniques gives information about the capabilities of neural network techniques:

- a neural network with delay is equivalent to a standard filter
- Hopfield nets are similar to stochastic multi-variable optimisation and to spin glasses

3.2 Design Decisions

On a more practical level there are serious problems that have to be considered during the development of connectionist systems. It is important that the techniques and parameters chosen during the design phase are appropriate for the task to be performed by the network.

Relevant input, a network structure that is appropriate for the problem, representative training examples, and sufficient training all have to work together to enable a network to learn to perform a task. Without these a net may perform badly or fail to learn the given task altogether. It is therefore necessary to know quite a lot about the problem and its domain in order to make the right choices and provide sufficient information to the system, because these decisions depend a lot on the particular problem and its domain. This is quite contrary to the common notion that neural nets 'program themselves' and therefore need no careful design. The major difficulty is that there are few definite rules to guide the developer in this task.

3.2.1 Representation

The first problem is that of knowledge representation: how are input units to be used in order to represent important features of the task domain, and what are these relevant features.

It is important that the network is given all *relevant features*. If any are missed out then the network must base its decisions on insufficient information which leads to poor decisions and therefore to poor performance. However, neural networks are good at dealing with the opposite problem where some of the information given is irrelevant. In some domains the problem of which features to represent does not arise. The net is simply given all information that is available. Examples are processing historical data and processing sensory input like visual images.

It has been suggested that neural networks can be used for forecasting and prediction of future events or developments. One financially interesting example is predicting prices in the stock market through training a network with a history of prices. The network should learn the behaviour of prices and predict future prices on the strength of it. However, it is known that prices on the stock market depend heavily on social and economic events which are hard to represent and hard to predict themselves. Without access to such relevant information it must be impossible for a network to find ways of solving this problem correctly [Whi88]. Improvements on other forecasting techniques may be gained but without sufficient causal support.

The actual *representation* of the relevant features is important too. The most basic decision on how to represent relevant information is the choice between local and distributed representations (see section 2.1.3).

It can also help a network if features which are known to be important are represented with more detail or are otherwise emphasised. This way the relative importance of features can be conveyed to the network as well as the features themselves, and variations in important features will have more drastic effects than variations in less significant features.

Representation has been identified as an important and difficult issue in other fields, like AI. However, the problems of representation for neural networks is more pronounced because the features to be represented are micro-features as opposed to the macro-features used in symbolic AI. This makes the representations harder to interpret (see below) and thus it is less intuitive to determine whether an appropriate representation has been found.

3.2.2 Network Architecture

The different network types described in section 2.3 are suitable for different problems. Usually the decision of which network type to use is not too difficult. But even at this stage wrong decisions are made, such as choosing a network with hidden layers when a simpler architecture is sufficient. It depends on the type of problem whether or not layers of hidden units are needed and it depends on the input space how many hidden units are needed.

A more difficult problem is to choose an appropriate configuration: the right number of units and the right connections between units. Usually deciding on the number (and task) of input and output units is relatively easy compared to hidden units: how many hidden layers should be used and how many units per layer? If too few hidden units are used the network will not converge, if too many hidden units are used the network will be able to memorise the training examples rather than learning from them. In both cases the network will not be able to perform the desired task correctly.

One approach is to determine which architecture and configuration is theoretically *sufficient* to perform the given task. In some cases this is possible. However, in many cases this is not enough. A network may theoretically be able to learn how to perform a task, but not practically. This is due to the fact that many real world problems stretch the practical limitations of neural nets when it comes to training time and operating time, and requirements for hardware performance may be too great. Therefore it is often necessary to determine which architecture and configuration fits a task *best*.

3.2.3 Training the Network

A problem related to input representation is choosing the set of training examples. It is important for the performance of the net that the training examples are representative of the problem domain. This is due to the ability of neural nets to detect regularities in input sets. This ability will not only detect features of inputs that are common due to the nature of the domain, but it will also detect common features that are due to coincidences. The network cannot distinguish between these two different kinds of regularities. A frequently cited example is the case where a network was to learn to distinguish between photographs with tanks in them and those without. The example photographs were presented to the net and the net learned to distinguish between them. What the developers failed to recognise was the fact that all photographs with tanks in them were taken on sunny days whereas all those without tanks were taken on cloudy days. The network *did* recognise this and it based its decisions on the overall amount of light in the photograph rather than on the images themselves. This is a good example of how difficult it can be to provide a good set of training examples. The problem here is the fact that the network is not told what to base its decision on – that is precisely what the network is supposed to learn. All the network is given is example decisions, not the reasons for the decisions. This shows that although the ability of neural nets to detect common features is a powerful technique, it must be used with care.

Again, a good understanding of the task domain is needed to be able to select a representative set of training examples. This understanding is also needed to determine which problem the net actually solves. As we have seen it is easy to miss regularities in training examples. The net then solves a different problem from the one it was expected to solve – in the example above the network solves the task of detecting sunshine when it was supposed to detect tanks. The problem is not that the network was unable to learn the task but that the developers were unable to specify the task unambiguously to the network. It is important to remember that networks have no concept of useful tasks and therefore have very different criteria in deciding which input features are most useful to make a decision.

Usually there is a choice of learning algorithms that can be used for a given network architecture and configuration. During the training phase the learning algorithm used can have a drastic effect on how fast the net learns. It can also affect how well the network learns its task, and thus influence performance during operation as well as during learning.

Not only the learning algorithm itself is important, but most algorithms use parameters that can be set by the developer to influence learning behaviour and performance. In some cases the influence these parameters have on training times are impressive. They may determine the probability of the net getting stuck during training, i.e. failing to learn altogether.

There is also a more general problem of learning in neural networks. Most learning algorithms require training examples to be presented to the network a lot of times before the network has learned to solve the task reliably. However, observation of animals and humans shows that nervous systems can learn with many fewer presentations. For example, children sometimes learn words after only a single presentation. Neural network research in this area would benefit from cooperation with the fields of cognitive science, psychology and neuro-physiology in order to determine what happens and why.

3.3 Operation

On a more fundamental level there are problems that most neural network systems have which cannot be solved by making the right choices during system development.

3.3.1 Interpretation

Conventional AI systems work by manipulating symbols. It is therefore possible to follow their operation step by step and intermediate results are often just as meaningful as final results. The operation of such systems is comprehensible for the observer, so it is possible to determine how the system solves its task, whether it solves it without fault, and what parts of the system are responsible for errors. Looking at neural networks it is difficult to see what they do and how they perform their tasks. Neural networks are based on sub-symbolic representations which are difficult for people to understand at a glance. In most cases it is possible to process the inputs and outputs of networks so that it is easier to see what they mean. In supervised learning systems the input and output usually have meaning to users. They may, for example, encode symptoms, diagnoses and treatments. However, in systems that learn unsupervised directly from the environment (e.g. through sensors) even the network's input may be incomprehensible. Neural networks also generate their own internal representations of the problem to be solved, and they determine their own way of solving it. These network internal representations are extremely hard to interpret, especially in networks of a size needed to solve real world problems. Therefore the actual operation of the network and its problem solving process is inaccessible.

It is difficult even in non-connectionist systems to determine whether a program solves its task without error, although it is possible to use flow charts to model the system behaviour in detail and intermediate states can be examined. In neural networks it is a non trivial task to find out whether errors exist and determining the cause of an error is much more difficult. Empirical observation is often the only way to find out what a network is doing: giving the net symptomatic inputs and observing the outputs generated by the network can give information about the system.[3]

The problem of interpreting what a network does is enhanced further by the fact that there are often several ways to solve a problem. Therefore neural nets can generate different internal representations for solving the same problem. Two training sessions with the same training examples are likely to produce different networks which both perform the given task.

One problem of interpretation has been addressed in the previous section: interpreting which task the net solves (see also section 3.2.3). When networks are trained to perform a task they are not usually told what to base decisions on, therefore they may choose correlations that are not intentional.

It is not only difficult to observe a networks operation, but it is also difficult to interpret these observations. For example, during training it is hard to tell when a network has learned its task sufficiently well or whether more training will improve the network. It can also be difficult to see whether a network is stuck in an unacceptable state where more training will not give improvements or whether the network is just learning slowly. If a network *has* failed to learn its task this can be for a variety of reasons, e.g. the network may be stuck and the next time it will learn successfully, the choice of network type or topology may not be suitable for the problem, units may be of the wrong type, etc.

[3] this problem is well known in the field of behavioral sciences where conclusions are drawn by watching a 'system' in operation

3.3.2 Performance Assessment

A frequent criticism of knowledge based systems is that they use heuristics, not hard and fast instructions, and therefore it is hard to assess their solutions. This problem is enhanced for neural network systems. As it is difficult to determine what is going on inside a neural network, the need for justification of a network's result is stronger. Unfortunately, it is difficult to interpret neural networks' results and operation which makes it more difficult to explain solutions and network behaviour.

Comparing the performance of different neural networks techniques is difficult because there are few benchmarks and they can be used differently. Comparing neural network solutions to traditional statistics and other software solutions is also difficult for different reasons:

- neural networks have a learning phase which may continue during operation so that the line between programming and system usage becomes blurred;
- the problems being solved may not be comparable although the original tasks are the same. A neural network may use much simpler strategies than a programmer would implement.
- resulting systems will differ in flexibility, integration and similar aspects.

The benefit of neural networks can sometimes be seen only in those cases where other approaches have been attempted without success. This is not a satisfactory basis for assessing the comparative performance of solutions.

3.3.3 Scale

One of the most serious problems of neural networks is that they become large for real world problems. Often small illustration tasks are good examples for certain problem areas like classification or optimisation. However, when the techniques are applied seriously, implementation practicalities like space and time constraints can prevent successful solutions. Connectionist techniques are often infeasible because they require capacity beyond today's hardware. The problem of scale is not only that networks become large and therefore there are more updates to be done, but also that most connectionist learning algorithms perform badly on large networks, so the problems of scaling up on network size are compounded.

Additionally as networks get larger it becomes more difficult to find appropriate training examples and training itself becomes more difficult. It also becomes harder to assess a networks performance, verify that a network has learned to solve a task successfully, and it is harder to determine what large networks do.

Part of the reason for this scaling problem is the generality of neural networks. A large and richly interconnected network can learn a complex task, but it is equally well suited to learning many other tasks that are similar. This means that both the structure and the learning process of the network are not problem specific but more general purpose. Thus a large part of the development of a connectionist system does not make use of the particular characteristics of the problem to be solved. This is desirable for illustration purposes during the development of a technique. It shows the theoretical power of the technique and the range of possibilities. However, it can hinder the implementation of real world applications.

3.4 Comments and Solutions

Often criticism of one particular technique used in neural networks is used as an argument against the whole field. For example the fact that backpropagation is slow and performs badly on large networks has often been applied to neural networks in general, even though there are alternative learning algorithms that solve some of the problems that backpropagation has.

Other problems mentioned above only apply to specific network structures or specific problems. For example, the issues connected with input representation are less relevant for problems where sensory real world input is used. Note also that in some cases representation and training issues can be solved by topology considerations using different network architectures, structures or connectivity.

There are a lot of cases where strengths of neural networks present themselves as problems because they are used inappropriately. One such example is the detection of regularities which can present itself as a problem when coincidental regularities are being found. Used appropriately, this behaviour of neural nets is very useful. For some problems it is unknown what information is best used as a basis for decisions. In other cases existing correlations are not known to be relevant.

Note that a lot of the problems mentioned above are not particular to neural networks. For example the problem of representative examples is well known in statistical evaluations and in scientific experiments in general. If there are more free parameters than the developers expected, then the “experiment” cannot be used to gain any insights.

Neural networks are slow on large problems. This is a serious problem, but it has led to the development of modular networks and alternative network structures like cascade correlation. There are approaches where networks are trained on small tasks and then made to cooperate with other small networks to solve a larger task. Concepts like hierarchies and clusters are being used. Learning algorithms have also been improved significantly.

Neural networks are used as general purpose problem solvers. It is expected that a network can be used to solve a large number of different tasks as long as it is large enough. The concept of general purpose problem solvers is attractive, but not realistic. It seems that neural networks could learn from AI where attempts to develop general purpose problem solvers have failed and given way to the more successful development of domain specific knowledge based systems. Neural networks that have clearly defined tasks can be useful even though they perform simple functions. The scaling problem of such modest networks is much less severe than in more ambitious networks: it is easier to implement a large simple network than a complex one.

3.5 Summary

the problems stated above are:

- lack of theoretical foundations,
- selection and representation of inputs,
- network architecture,
- representative training examples,
- selection of an appropriate training algorithm,
- selection of learning parameters,
- interpretation of results,
- interpretation of operation,
- confidence in solutions,
- comparative assessment of performance,
- scaling.

The main problems that are not solved easily are:

- The theoretical foundations of neural networks are far from being satisfactory. It is still not known what the real limitations of neural networks are. This problem reduces confidence in applications.
- For neural network applications it is essential to know a lot about a problem and its domain: relevant inputs and representative training examples have to be used. This does not mean that it is necessary to know how to solve the problem, but it must be known what is needed to solve the problem in terms of suitable network structure, sufficient input information, and appropriate training with representative training data.

- Neural networks do not lend themselves to explaining their results and their operation in terms that people can understand.

Neural networks are still a research field. Progress is being made in determining their scope and limitations and overcoming their problems. Neural networks should be applied to problems that they are known to be good for. All other areas should be considered experimental until it becomes known whether they are suitable or not.

4 Strengths of Neural Networks

A lot of the criticism aimed at Neural Networks is justified. However, neural networks have been used successfully to solve problems.

In the following, different characteristics of Neural Networks are discussed. Some of these characteristics enable Neural Networks to support in a natural way certain aspects of problem solving which are difficult to handle with conventional techniques. There are cases in which neural networks provide solutions where other techniques have failed. Other aspects of problem solving do not match the characteristics of Neural Networks well and thus conventional approaches will be more successful. It is important to see both the strengths and the limitations of Neural Networks in order to determine what kinds of problems they can deal with effectively, and to outline the pitfalls of this technology.

Note however that in this report the approach taken to assess neural networks is pragmatic. The issue is to establish their potential for applications rather than their formal, theoretical potential. It is of little use for applications if a problem can theoretically be solved with neural networks but it is not practically feasible. On the other hand it is not important whether neural networks can solve all possible cases of a particular problem type as long as all cases it will encounter can be solved.

In this section the general strengths of neural networks are outlined, while the next two sections discuss how neural networks can be used for practical applications.

4.1 Generality

Neural networks use uniform methods for a wide range of problems. They are learning systems and which problem they solve depends on what they are taught. Most parts of the development of neural networks are to some extent problem independent. Thus the same considerations apply to a variety of applications.

Each part of a network is simple. Only through interconnection can the parts of a network solve problems of any interest. The units are simple processors; they have simple activation functions. The connections between individual units are simple and communication between units is low level which makes the communication protocol simple. This simplicity makes it easy to implement neural networks in hardware as well as in software. The low level representation and computation of neural networks makes them suited to low level sensory tasks.

4.2 Adaptivity

Neural networks are learning systems, potentially applicable to many problems. Neural networks learn to produce a response for a given input. When the network is confronted with new inputs from the environment, it can learn to produce the right responses for those inputs. At the same time other responses may become dated because the inputs are no longer present in the environment. The old input-response mapping in the net will fade with disuse while new mappings are established. The memory of solving a problem that never occurs will fade in a network that continues to learn. This behaviour enables networks to adapt to a changing environment.

A neural network can therefore be placed into an environment with a number of basic skills or instincts. Taking its inputs directly from the environment via sensors and learning through some form of feedback from the environment, the net can then pick up more skills that are useful for it to perform its task. This produces an autonomous system that can perform well in a complex changing environment without being re-programmed. This is extremely useful in situations where re-programming is not feasible (e.g. systems in space).

4.3 Feature Selection

Learning in most neural networks is based on Hebbian learning which uses activity correlations between connected neurons. This enables neural networks to detect correlations between input features and problem solutions. They can find significant criteria in their input which they can base decisions on.

The ability to detect correlations is important for deciding which solution to choose. This can be used in a variety of ways. It is not only a way to find connections between inputs and desired outputs, it can also be used for classification purposes. Consistent and relevant features can be identified, irrelevant information and noise can be filtered out leaving only useful information for problem solving. In this way networks can pick out those features that are most effective for separating different classes. Networks can thus be trained to improve discrimination to makes classification and recognition more effective. Existing degrees of freedom in the input can be determined which is necessary for dimensionality reduction. A network can learn to use available dimensions to represent different input patterns.

The classification and generalisation abilities of neural networks are based on nearest neighbour strategies: similar things have a similar representation and similar output. This makes problems like XOR hard to solve because here generalisation is punished: the nearest neighbour must give a different response.

4.4 Tolerance

Representation, problem solving knowledge and information processing can all be distributed in neural networks. That means that each component contributes a small part to the whole system which reduces the importance of individual components. Together with the general flexibility of neural networks this makes neural networks tolerant against damage or loss of components, missing information and noisy data. If a unit is damaged or destroyed a neural network will continue to operate. The performance of the network will degrade gracefully rather than fail immediately, because the whole network depends on each unit only for a small part of its operation and other units can take over from a damaged one. Thus the network continues to operate using the available processing power. In the case of missing information in inputs during operation, a network will use the available information to produce its results. Again the degradation is graceful. In both cases the results may be less reliable or of lower quality, but the network will continue to produce the best solution possible. It is remarkable how well networks can perform in the face of missing components or missing information.

Neural networks are also tolerant against noise in input data. The most prominent characteristic of noise is that it cannot be tied to input features. It is a distortion that affects all parts of the input in the same random way. Noise does not produce any correlations and therefore a neural network will ignore it. Neural networks can be used for learning in the presence of noise. In some cases noisy inputs help the net to identify relevant features and therefore will make learning and generalisation more effective. Neural networks can also be used to eliminate noise, i.e. they can clean up noisy patterns.

4.5 Imprecision

Neural networks are pragmatic in their approach to problem solving. Often they cannot reliably produce the best possible solution to a problem. However, in many problem areas they can provide a good solution fast where a perfect solution not available (missing or noisy data), or too time-consuming to find. This is an important capability when it comes to applications. In the real world, time is often a crucial factor and quality is often only vital where solutions are found easily. Thus the pragmatic approach of neural networks gives them an advantage in real-time systems and for complex problems that are difficult to solve.

There are different types of imprecision. There can be vagueness: how precise is the task - is a margin of error acceptable? Or there can be uncertainty in a degree of belief: how much confidence is there in data and processes? Fuzzy systems deal with vagueness by using varying degrees of applicability rather than the yes/no decisions of other systems. This philosophy is close to neural networks. Finding good matches, not just perfect matches, is an example of fuzzy matching; using fuzzy set theory a pattern can be classified in different ways, having different degrees of membership for different classes. This can be implemented in a natural way using continuous valued output units for different classes.

Dealing with uncertain data, the same mechanisms can be applied. Neural networks are well suited to handle uncertainty because of the way information is propagated through the network. Decisions are taken locally based on information available. However, these are good decisions rather than the best decision, and therefore no commitment is made at these intermediate stages. The network can postpone any such commitments until the end of the computation when all relevant information has been taken into account and a decision can be made at that level. Until then all good alternatives can be considered. The confidence in the result can be expressed using the network's output values in the same way that different class memberships are handled.

4.6 Self Programming

Neural networks are not programmed *how* to do things, but they are taught *what* to do. This can make system development easier if there is little or no information about the problem solving process. Neural networks can be implemented with information about the problem itself.

Neural networks have been applied in cases where other techniques have failed. In some cases this was due to the fact that programmers failed to identify those features of the information available that are most suitable for determining the correct behaviour. One such example is ALVINN [Pom89b], the autonomous vehicle. Here programmers were over-complicating the problem of keeping a vehicle on a road, and therefore they could not provide a viable solution. The neural network implementation did learn to stay on roads and it could perform this task in real time more reliably and faster than the AI based implementation. The neural network may have found a sub-optimal way of solving the problem, but it was good enough and could be improved.

4.7 Alternative View

Connectionist approaches to problems are very different from other approaches. This provides an alternative angle from which problems can be viewed. This alternative angle can make solutions to some problems more obvious because it makes it easier to think about the problem. The way of thinking about a problem determines how easy it is to find a solution. Once a solution has been found it may be appropriate to implement it using a different paradigm, which does not make the initial approach less valuable.

Neural networks have a strong non-deterministic aspect which can produce unexpected or unplanned solutions. This is not only a disadvantage that reduces confidence in solutions, but it can also add a chaotic element to the system which may result in creative behaviour.

4.8 Physiological Analogy

Another advantage of neural networks is that ultimately they are based on the nervous system. The brain is an example of a connectionist system that works. To an extent this proves that it *can* be done. Admittedly there is a big difference between today's connectionist systems and the brain, but there are similar concepts involved. Not all connectionist research is physiologically plausible as the motivation is often to produce a working system, not a plausible one. However, the fields of neural networks and neuro-physiology are related closely enough for both to benefit from the relationship. Connectionism draws ideas from nervous systems and it has provided physiologists with ideas through exploring possible implications of physiological findings. Neural network models can be used to explain or interpret experimental data. Assumptions made in neural network research can be verified with physiological research.

The brain is capable of producing solutions to many of the problems that the AI community is trying to solve with the help of computers. As physiologists find out more about how nervous systems work, connectionist systems may become more plausible and more powerful.

The brain is not only an example of a connectionist system that works, but it is also an optimised system. All biological components are costly for the organism, both in terms of an individual's energy and in terms of the species' development. This makes the nervous system even more attractive as a model for software and hardware systems.

4.9 Summary

The main strengths of neural networks are:

- generality: uniform methods for a variety of tasks;
- simplicity;
- adaptivity: learn and possibly forget and thus adapt to changing requirements;
- correlation detection: association, classification;
- feature selection: find most relevant features, improve discrimination, generalisation;
- tolerance against damage: processing performance degrades gracefully;
- ability to handle missing information: results degrade gracefully;
- ability to be used with noisy data in training and operation;
- pragmatism: find a good solution fast, not the best solution slow or never;

- good at fuzzy matching: allow a margin of error;
- deal well with uncertainty: reflect belief;
- self programming: find problem solving process;
- alternative view: make it easier to think about a problem;
- connection to physiology.

5 Applications of Neural Networks

This section concentrates on the application of neural networks. The strengths and weaknesses discussed above indicate application areas to which neural networks are suited. Applications to which traditional AI techniques have been applied are of particular interest in this report and are dealt with separately in the next section. For a more comprehensive review of applications see, for example, [WS90].

5.1 Comments on Current Applications

Currently neural networks are being applied in many different areas. In order to establish the suitability of neural networks to different areas it is important to look at the motivation of implementations as well as the results. Many applications in neural networks are driven by the need to determine what neural nets are capable of. There is little theoretical foundation for the more powerful connectionist techniques and therefore work is being done in order to find (and extend) limitations. Often these applications are pathological rather than sensible. For example, the fact that neural networks can solve the XOR problem is not important as an application for neural networks, but rather as a step in establishing limitations. Systems that are incapable of solving the XOR problem are seriously limited in their applications because many problems require that kind of problem solving ability.

There are misunderstandings due to the current popularity of neural networks. The expectations on neural networks are as high as the interest, therefore there is considerable pressure on the field to produce results. This way many applications that were not intended to be used in the real world are taken to be working systems. This makes it difficult to single out those cases where neural networks *have* been successfully applied to real world problems in situations where they provide a solution that was previously unobtainable.

On the other hand there are many people who want to jump on the bandwagon, promising solutions without sufficient knowledge of the field. This leads to implementation attempts in areas where neural networks are not the best way to solve problems and more conventional techniques would be more appropriate and more efficient.

The pressure on neural networks to produce solutions was applied at a time when the field was still very much a research area, which indeed it still is. Results were required when theoretical applicability questions had not been answered. It is no surprise that this has lead to disappointments. However, neural networks do provide new ways of solving problems and they can be used successfully in real world applications.

5.2 Application Areas

There are two major groups of applications that use neural networks in different ways:

1. storage: if a neural network is used to store patterns it can be used to reproduce these patterns during network operation. This way networks can be used for tasks like eliminating noise from input patterns, completing partial patterns, encoding and decoding patterns, data compression and for generalisations by returning the closest stored pattern for a given input pattern. Associative storage with neural networks is well understood.

2. problem solving: the network is taught to solve a problem by associating input patterns and correct responses. This way networks can be used for mapping, signalling, classification, control, etc. Strictly speaking, storage is a specialisation of this group of applications because the correct response is to reproduce the original input pattern.

Neural networks operate on the sub-symbolic level which makes them suitable for low level sensory tasks like visual image processing (character recognition etc.) and low level speech processing (recognising words from auditory input). Their tolerance to noise and imprecision in data makes them especially suitable for processing sensory inputs taken from the real world. The main problem in traditional systems that try to perform sensory tasks is that the real world is not precise and predictable enough for the techniques available. Processing low level signals like radar signals, ECGs, seismic lines and similar measurements can be done with neural networks. In these applications there is typically a lot of data with some exceptional features which can be used for classifying the signals or which require further attention. A network can single out abnormal or interesting data from the bulk of observational data.

Neural networks can be used to digest large amounts of (complicated) data finding the relevant features and regularities of the data and thus condensing the data. Note, however, that there are other, more conventional techniques that can be used for data compression without loss of information. Neural networks may prove to be useful in areas where some loss of information is acceptable in order to achieve larger compression factors. Neural networks can be used to learn which data is relevant and must not be discarded (e.g. [El91]). Neural networks provide appropriate techniques where discrimination enhancement, feature extraction or generalisation is required.

Finding good matches fast makes neural networks suitable for complex optimisation problems and constraint satisfaction problems. If there are many good solutions, a neural network can reliably produce one of them, often faster than alternative techniques. Problems like load balancing, circuit layout and routing problems (e.g. traveling salesman) have been solved successfully with neural networks. A benefit of neural networks in constraint satisfaction is that a network will produce a sensible solution even if not all constraints can be satisfied.

The fuzzy nature of neural networks and their tolerance to noise are distinct benefits. In many applications these properties are desirable because precise information is not available or not appropriate. For example in constraint satisfaction problems the constraints may be vague, requiring fuzzy matching between real world states and constraints.

Process control like motor control for robots and regulating industrial systems have also been performed well by neural networks. Being trained by observation, networks often make a better job of modeling the systems to be controlled, and thus can produce more appropriate regulating signals.

Neural networks are good for adaptive pattern classification. They are easier to train and to use than traditional pattern classifiers which makes them more practical.

In section 3, prediction of stock market prices was given as an example of a problematic application because the relevant information is not available. Other areas of forecasting present a different situation. In some cases there is more evidence that relevant information is available even though it may be unknown what part of the available information has impact on developments. In these cases neural networks may well be able to predict future events.

The ability of neural networks to find inherent regularities or laws is double edged. In some cases good results can be achieved and there have been successful applications. However, it is not clear which problems of this kind can be solved and which cannot. Therefore assessment of such systems is empirical, and the success of solutions depends much on whether sufficient information and good training data is available.

Trying to solve tasks that are themselves poorly understood does not increase the probability of success. A net can only find relevant features in the data provided. If there is not enough information in the data (if relevant features are missed out) the net will not be able to extract useful regularities for its task and thus will fail to deliver the required behaviour. A major problem is that the net may perform well on training data, but when presented with new data it will show unexpected behaviour. This can be caused by an inappropriate network architecture which allows the net to memorise training examples instead of generalising, or by training data that contain coincidental regularities. It is difficult to rule out the possibility of a net relying on irrelevant features if the relevant features are not known.

5.3 Directions

There are various possibilities for how the problems of neural networks can be overcome so that their benefits can be used.

As mentioned above, one of the trends in neural network development is modularity. Modular systems where each module is small, although perhaps complicated, and modules work together to perform real world tasks are a way to overcome the scaling problems of neural networks. They can also be a framework to pull together experiences of different fields. Each small, special purpose network can perform, for example, a specific mental function. In [Min86] a "society of mind" is outlined using distributed cooperative processing with heterogeneous modules (see also the next section).

Different situations and domains require different approaches to problems: sometimes it is good to learn carefully, gathering experience. In other cases (e.g. if there is little time) it is better to make fast generalisations and act accordingly. These differences should be taken into account for neural network research and applications.

An indirect advantage can be taken of neural networks by trying to let them extract relevant features of data and then use the knowledge gained from the net's behaviour to represent the data appropriately in more conventional ways.

5.4 Some Important Applications

The Bomb Sniffer The Thermal Neutron Analysis system (TNA) developed at SAIC [SL89] is a security system to detect bombs in luggage. The system works by bombarding the luggage with gamma rays and analyzing the resulting emission from the luggage by matching them to emission patterns which plastic explosives give off. The neural network approach was tried after two years of development using statistical pattern recognition techniques failed to meet the imposed acceptance criteria. The neural network system met the criteria in less than three months. However, there were problems with the neural network approach, some of which were solved (e.g. the emission patterns of women's shoe heels are similar to those of plastic explosives!). However, the system is large, expensive and too slow to be used seriously in airport security.

Cascading Neural Networks In [Hil91] modularity is suggested as an approach to overcome the scaling problem of neural networks. Instead of developing a single network to perform an overall task, subtasks are identified together with their inputs and outputs (information needed to perform each subtask and information generated by performing it). Outputs of one subtask can be used as inputs for another which results in a cascade of subtasks which are connected via their inputs and outputs. For each identified subtask a training set is generated and a neural network is implemented that solves the subtask. Each of those networks is trained in isolation until it can perform its subtask. When each of the networks has been tested (on training data, anticipated real world cases and extreme cases), the networks are integrated resulting in a cascading neural network. This in turn is tested. There are two main advantages of identifying subtasks and developing networks to perform them: for each network it is easier to generate representative training data, and it is easier to test each network to determine whether it has learned its task.

Fault Diagnosis [BL91] A hierarchical (cascading) neural network is used to diagnose faults in chemical process plants. The first network looks at the overall process to determine where the fault occurred. The subsequent networks are more specialised: they only look at their particular part of the process to localise faults further. Thus the focus of the diagnosis is increased using more and more precise fault types. This strategy enables the network to degrade gracefully with fault situations that have not been encountered before. The network will produce useful information on where the fault is localised even if it cannot produce a complete, precise diagnosis.

Prediction An example of a successful prediction application is the prediction of secondary structure of globular proteins from amino acid sequences [QS88]. Proteins are chains that are made up of amino acids. The secondary structure of proteins is the local structure which is formed by chemical bonds between amino acids in the chain. The form of this structure is determined by the sequence of amino acids in the chain. There are about 21 different amino acids. The network's input is a window containing a sequence of 13 amino acids. This window is passed over the whole sequence of the protein. This input is represented as 21 mutually exclusive units (denoting different amino acids) for each of the 13 positions, i.e. 273 input units. The network has 3 output units which represent the secondary structures: α-helix, β-sheet and coil. The network is trained with a set of around 100 different proteins. The training examples were chosen to be dissimilar, whereas the testing examples were a more random selection of proteins to determine whether generalisation had been achieved. Initially networks with one layer of hidden units were used and trained with back-propagation. However, experiments showed that hidden units do not increase the quality of predictions. The networks used the hidden units to learn particular examples rather than contributing to generalisation desired. The prediction results of around 63% accuracy are good compared to previous approaches to prediction using different techniques like statistics.

Character Recognition Neural networks are frequently applied to the problem of character recognition. One interesting example is the work of LeCun et al. at AT&T Bell Laboratories [LeC89]. A network is trained to process original hand written zip code data of the US postal service. The data (written characters) to be processed is extremely noisy. A network with three hidden layers is used. The input consists of the image of a character represented by 256 units. There are 10 output units (one for each digit). The network is trained using back-propagation. The interesting part of this system is the way in which different layers are used. Ideas from Kohonen's feature maps are used together with a specific connectivity and weight sharing in order to make learning more efficient. Information about the domain is used to modify the architecture of the network so that it suits the task to be learned.

The first hidden layer is responsible for extracting local features from the input. There is a range of feature detectors for each small area (5 by 5) of the input image. This allows the network to detect features regardless of where they are on the input plane. The other two hidden layers are used to detect higher order features and finally to detect the digits themselves. This functionality is achieved by restricting connections between layers and their weights. Connecting to only few units from the same area forces units to use only local information. The same feature detectors are needed for different locations on the input plane. This is achieved by weight sharing: different units that detect the same features at different locations are forced to share weights. Note that weight sharing and eliminating connections makes learning by back-propagation more efficient.

5.5 Summary

General problem areas that neural networks are well suited for are:

- problems in areas where finding a good solution fast is preferable to finding the best solution slowly;
- problems where a lot of data must be processed fast (condense data);
- processing (evaluating) large amounts of historical data;
- problems with a high likelihood of solution in results;
- associative storage applications like eliminating noise and pattern completion;
- low level sensory tasks (vision, speech);
- low level pattern recognition (detect targets, recognise letters);
- low level signal processing (radar, ECG, seismic lines);
- feature extraction;
- contrast enhancement;
- classification tasks;
- generalisation tasks;
- optimisation (circuit layout, load balancing, routing);
- constraint satisfaction;
- process control (regulating industrial systems, motor control);
- pattern matching, mapping;
- prediction.

Development directions are:

- modular neural networks;

- special purpose networks;
- special purpose learning strategies;
- heterogeneous systems.

Example application areas are:

- signal processing: the bomb sniffer;
- fault diagnosis in chemical plants;
- Prediction: protein structure;
- character recognition: zip code data.

6 Neural Networks and AI

In this section current applications of neural networks within (traditional) AI are described, and possibilities are shown of how the two fields can benefit from each other. In the past neural networks have been seen as an alternative to AI, but people are beginning to realize that they can complement each other in many ways:

- AI systems can use neural networks for learning
- AI and neural networks can be used alongside each other in hybrid systems
- AI can be used to build neural networks
- neural networks research can learn from AI. Some of the problems and criticisms are the same in both fields

The applications in sections 6.1 and 6.2 show how some of these approaches can be used in system development. Systems that combine AI and neural networks are dealt with separately in section 6.2 because they are especially interesting. The different approaches are discussed further in section 6.3.

6.1 Some Applications

The applications in this section show some attempts at combining AI and neural networks techniques, or applying neural networks techniques to traditional AI problems.

6.1.1 Neural Networks as Knowledge Based Systems

Early attempts to use neural networks for AI tasks led to implementing whole expert systems or knowledge based systems using neural networks instead of the traditional AI techniques. These implementations show some of the formal power of neural networks. However, it is not clear that they provide improvements on more conventional implementations of AI systems.

Neural Networks for Production Systems Touretzky and Hinton [TH86] used neural networks to simulate a production system. Their Distributed Connectionist Production System (DCPS) was developed as a feasibility study for using neural networks to implement one of the basic architectures of knowledge based systems. The advantage of the resulting neural net is its ability to find the best match in the absence of an exact match, and its robustness against damage and noise.

A *production system* stores facts and rules in its knowledge base. A working memory is used for storing facts of a given problem. Production systems work by applying rules

whose preconditions match the facts in the working memory. Applying rules updates the working memory by adding or deleting facts. The DCPS is restricted to simple facts which are triples such as (ADE). The rules are of the form
(=xAB) (=xGB) :- +(G=xP) -(=xRB)
where '=x' stands for a variable, + and - stand for addition/deletion to the working memory. Thus the above rule says: if the working memory contains two facts with the same (unspecified) beginning x, one of them ending in AB and the other one ending in GB then add a fact starting with G ending in P with the same x in the middle to the working memory and delete a fact ending in RB from the working memory.

In DCPS *facts* are stored using distributed representation: each fact is represented by several units. Each unit is used to represent parts of many different facts. However, the overlap of units between representations of different facts is very small (usually not more than one). Thus the representations are tolerant against noise and damage.

Rules are stored in a semi-distributed way. Each rule is represented by a clique of about 40 units. These cliques do not overlap. The rule store operates in a winner-takes-all mode so that only one rule is active at any time.

The problem of finding the rule whose preconditions best match the current working memory is interpreted as a constraint satisfaction problem. The energy of the system's state is a measure of constraint violation. Thus in a state of minimum energy a maximum number of constraints are satisfied. The constraint satisfaction is done by simulated annealing on a Boltzmann machine.

Expert systems on Neural Architecture [Fu90] In this application expert system components are mapped onto neural networks in the following way:

- Knowledge representation: rules are implemented as connections and their weights. Premises and conclusions of rules are represented by units, the strength of the rules (degree of belief) are reflected in the connections' weights. Predicates are represented using units' thresholds.
- Knowledge processing: the symbol generation and pattern matching of expert systems is turned into combining and propagating activations in the neural network.

As an example consider the following rule in a medical diagnosis system:
if the culture's site is the throat and the organism is identified as Streptococcus then conclude that the organism belongs to the subtype Group-D-800.
This rule would be implemented by three units, one representing that the site of the culture is the throat, the second representing that the organism is a Streptococcus and the third representing the conclusion. There are connections from both premises to the conclusion. The weights on the connections are sufficiently high so that the activation of both premise units will result in the activation of the conclusion unit.

6.1.2 Neural Networks and Language

There are neural network applications in both understanding and synthesis of natural language. The following is an example of speech synthesis: transforming encoded text into phonemes.

NETtalk [SR86] Sejnowski's NETtalk is one of the first demonstrations of a serious size neural network using several hundred units. A net with one hidden layer was trained using back-propagation. The net is presented with strings of encoded English text. The task of the net is to produce input for a speech synthesizer (phonemes). The net was analyzed by recording the states of the hidden units for each input/output pair. It was found that the net generalised very well - letters that produce similar sounds were mapped onto similar representations. Thus the NETtalk could perform well on new words (which had not been used for training), 'pronouncing' them similarly to the appropriate known words.

6.1.3 Neural Networks for Robotics

Robotics is one of the traditional fields of AI. The use of neural networks for robotics has been investigated mostly as alternative rather than additional techniques that can be integrated.

The classical approach to robot control is either to model the system explicitly or to teach the robot. In the example of controlling a robot's arm the system can be taught by moving the robot's arm from the current position to the desired position. The system records the move and it can later reproduce it. Note, however, that only these pre-recorded movements can be performed. The other main traditional approach is for an analyst to model the robot's arm explicitly. With the help of this model, the geometric constraints involved in transferring the current position into the desired position are used to generate the control sequences which are needed to perform the move. Accurate models are complex.

In the neural network approach a model of the robot is created by the net itself through learning. Only those features of the robot are modeled which are relevant to the tasks it has to perform which often results in simpler models. The behaviour of a robot controlled by a neural net is flexible. The robot can apply learned behaviour to new, unforeseen circumstances. Patterns that are not used for a long time cease to be represented in a neural net. Thus the robot can 'forget' learned behaviour if it is no longer relevant.

In the area of autonomous navigation neural networks can be used to allow a robot to form 'its own' representation of its environment. In conventional systems the robot is provided with maps that are designed and organised by humans, who have their own ideas about appropriateness of different forms of maps (e.g. spatial vs. procedural). A robot provided with a neural net can represent its environment according to the features that are relevant to the robot's task. There are two main advantages in this approach. The first is that the robot can perform better: once the relevant information is found the required information can be readily extracted. The second advantage is that the robot can adapt to changes of the environment by replacing features that are no longer important with new relevant ones.

ALVINN – Autonomous Land Vehicle In a Neural Network ALVINN [Pom89a, Pom89b] is a good example of an application where neural networks provided a solution to a problem where traditional approaches have failed. The problem to be solved is to drive a van. The original approach requires the programmer to identify relevant features, implement detectors for them and implement the transformation of these features into steering instructions. The resulting system required lots of programming time and it was ineffective and reliability could only be achieved at slow driving speed. Additionally the resulting system was very problem specific so that re-programming was required for driving on different road types. The neural network approach achieved twice the speed of driving with the same reliability. It was easier and faster to implement and the same basic system can be used for different road types.

ALVINN is a network with one hidden layer. It learns by 'watching' a person drive for around 5 minutes. Back-propagation is used to determine which features in the input image (an image of the road ahead) are most correlated with correct steering direction. It was taught to drive on single lane paved and unpaved roads and multi lane lined and unlined roads.

6.2 Combining AI and Neural Networks

The approach of neural networks and traditional AI (and other fields too) to solving problems is quite different. The fields use different techniques with different motivations, and they have different strengths and limitations. Hybrid systems provide an environment where different parts of a problem can be solved by different techniques such that each technique is brought into play where it is best suited. This section concentrates on approaches that have taken advantage of the differences in AI and neural network approaches to some extent.

6.2.1 Cooperative Problem Solving

In cooperative problem solving modules or agents are used to perform sub-tasks of the overall problem solving task. Each agent is a special purpose problem solver, performing a small part of the system's overall functionality. Different agents can use different problem solving techniques. Neural networks can be used as agents to perform tasks they are suited to.

FRESCO – FedeRative Expert System COoperation In [DSS92] federative systems are suggested as a framework for autonomous agents that can cooperate to solve problems. A system has been implemented in which agents bid for current tasks and the most competent agent is selected to produce a result. FRESCO, a federative knowledge based system, is extended to incorporate neural network agents ("neural problem solvers"). The neural network components are wrapped in translation modules which overcome the gap between symbolic representation used in knowledge based systems and sub-symbolic representation of neural networks.

One interesting part of the work described is a classification of neural networks based on their development and usage. Three dimensions of flexibility are used for classification: network development, network operation and the problem to be solved itself.

1. A distinction is made between *ad hoc* networks which are built especially to solve a problem instance, and stable networks which can be used for several problems of the same type.
2. Stable networks are further divided into static networks which complete their learning phase before they are used, and dynamic systems which continue to learn during operation.
3. Finally, dynamic systems can continue to learn in order to get better at solving the same problem (stable requirements), or they learn to be able to adapt to a changing problem (dynamic requirements).

Different network classes require different methods for incorporation as network components into the federative framework. Note, however, that in principle there are no restrictions on network architecture or the learning algorithms of the neural networks themselves.

The Neural-Symbolic Paradigm In [Khe92] an object-oriented approach is used to provide a framework similar to FRESCO. Objects represent information processing paradigms such as neural network simulators, expert system shells, etc. Each such object is derived from a generic interface object which is used for providing standard messages and functions as a uniform interface to other objects. Inheritance is used to facilitate the introduction of new paradigms. Suggested application areas for the Neural-Symbolic Paradigm are those that contain both explicitly known components and fuzzy, poorly understood components. Examples of such areas are finance, manufacturing and control, and natural language processing.

INNATE/QUALMS Diagnostic Performance Analysis [BLN91] describes how fault diagnosis can be improved by using both expert system and neural network techniques. The contribution of neural networks is mainly their ability to handle noisy sensory inputs fast, and to learn from experience, whereas the expert system contribution is their ability to generate explanations and easier maintenance due to explicit representation of knowledge. The system uses hierarchical neural networks (see also section 5.4) to diagnose faults based on sensory input like process measurements. The networks' diagnosis is analysed by a model-based expert system. Using deep knowledge, the expert system interprets the networks' output and determines whether to confirm the networks' diagnosis or to offer alternative solutions. For fault diagnosis in a chemical process plant the expert system contains knowledge about the plant's structure and function, which gives the system a context for the interpretation of neural network results.

Interactive Diagnostic Experts In [LLW92] neural networks and rule based techniques are used to capture different types of knowledge and to provide different mechanisms for solving different types of problems. Neural networks are used to capture implicit knowledge that can be gained from past experience whereas rules are used for explicit background knowledge. In trouble-shooting applications the neural network can diagnose common cases that have been seen before, which is a fast process. Cases that have not been seen before can be dealt with using the rule base for producing flow-chart type instructions that lead to a diagnosis step by step. This is a slow but sure approach. A case that has been solved using the rule base can then be added to the neural network's 'experience'. Applications of this framework are INSIDE and COINCIDE.
INSIDE (Inertial Navigation System Interactive Diagnostic Expert) supports the diagnosis of avionic equipment. The system is in operation at Singapore Airlines. The system is described in [LLTT91].
COINCIDE (COnnectionist INcremental-learning Crane Interactive Diagnostic Expert) is a prototype system developed for the Port of Singapore Authority. It is a system for teaching technicians to troubleshoot quay cranes.

6.2.2 Learning, Knowledge Refinement

Neural networks can be used during the knowledge acquisition phase of expert system development. The resulting neural network can become the system's knowledge base, or the network can be analyzed to extract a traditional knowledge base. The task of knowledge representation in AI systems can be aided by letting a neural network extract the relevant features from the data available. These relevant features can then be represented using AI's symbolic representation techniques. The problem here is that neural networks are hard to analyze.

Towell *et al* at the University of Wisconsin, [TSN90, TCS91], are looking into combining techniques from neural networks and expert systems to overcome some of the problems of the two fields. Neural networks lack explanation, have long learning phases and cannot use explicit *a priori* information about the problem solving domain. They may also use coincidental correlations in training data. On the other hand, expert systems cannot deal well with imperfections like incorrect or incomplete information in the knowledge base. [TCS91] suggest the use of neural networks to improve knowledge bases: the neural network is built using the knowledge base to determine its structure, connectivity and initial weights. The network is then trained on examples to refine existing rules. In the network structure, room is left for improvements allowing for missing information in the knowledge base. In this way new rules can be learned from the examples. When the network has finished learning, the new knowledge base is extracted from the improved (taught) network. Thus the knowledge base is corrected and completed.

Some of the problems of this approach are:

- there are severe restrictions as to what type of knowledge bases can be refined in this way,
- automatic naming of rules and extracted features is difficult,
- the resulting knowledge bases still need human interpretation,
- there is some amount of unformalised human intervention involved in the approach.

However, the approach allows neural networks to be used to determine problem solving processes that need not be explained, like vision and other low level processes. Low level rules that are not known explicitly can be extracted and used. A remaining problem is whether the extracted rules are causal or based on coincidents.

6.2.3 Process Control

Neural networks have been used successfully in process control. The following is an example of a neural network working alongside a rule based system.

Intelligent Arc Furnace^{TM}: [SS92] At North Star Steel - Iowa neural networks are used to control an electric arc furnace. Electric arc furnaces in steel production operate by using large electrodes to deliver power into scrap metal. The positioning of the electrodes is vital for effective operation. An electric arc furnace is a highly complex system. Its properties change during operation depending on many parameters, and different parts of the system interact. This makes it difficult to model the processes correctly. In addition, the response of a furnace to regulator signals is critical and has to be considered when the signals are generated. Reaction delays are costly and may be dangerous. These problems lead to difficulties in using traditional process control techniques, including rule based systems.

Three neural networks were built using over 200 units that are fully connected. The networks were trained using supervised learning with the extended delta-bar-delta rule ([Jac88] a speedup of backpropagation).

The first network is trained to emulate the furnace regulator, i.e. the furnace control system previously used. Previous and current regulator and furnace state values are used to determine the correct output for the regulator. The output of the previously used regulator is used to provide the correct solutions for teaching the net.

The second network is trained to emulate the furnace itself – it learns to predict the furnace's state changes depending on regulator settings. This network again uses the previous and current regulator and furnace state values as inputs, but additionally it is given the output determined by the regulator. The furnace emulator then determines the furnace's next state. The next state of the actual furnace is used to evaluate the network's solution.

When both these networks have been trained in isolation, they are combined to produce the furnace/regulator network. This network again receives the previous and current regulator and furnace state values. Both networks are used to produce the next furnace state. The desired furnace state is used to determine the error of the system, which may cause changes to the regulator part of the network.

The combined network is used to provide regulator settings for the actual furnace and to predict the furnace's response to regulator settings as an additional safeguard to avoid unwanted furnace states. This combined network improved the furnace's usage of power and its electrodes and it increased productivity.

An interesting aspect of this system is that the network is only used during normal furnace operation. In abnormal, critical situations a rule based system is invoked to provide control that ensures safety. This is an intriguing way of combining neural network techniques with traditional AI - the networks are used in normal operation where it is not necessary to know much about what is happening. As soon as something goes wrong, attention focuses on the regulation process and rules are used to determine the best action.

6.3 Directions

There are two major ways of combining neural networks and AI: the first is to use techniques from one field to support the other, the second is to use techniques from both fields to supplement each other.

6.3.1 Support

One possibility of using neural networks to support development of AI systems is to let a network learn the relevant domain knowledge during the knowledge acquisition phase of system development as described in section 6.2.2: once the network has learned the relevant domain knowledge it can be used as part of the knowledge base, or the network can be analyzed to gain insights into how the knowledge base should be implemented using more conventional (symbolic) techniques. Used on historical data neural networks can extract relevant features that can be used for classification, generalisation or other such tasks. This makes them suitable for pre-processing historical data or simply for determining which features should be represented to support the problem solving process.

On the other hand AI can be used to support the development of neural networks. The task of building a neural network to perform a given task is difficult and complex enough to be supported by an expert system that supports the developer in design decisions. Another potential application for expert systems in neural networks is for cascade-correlation networks (see section 2.3.7). When a unit is trained to perform a sub-problem an expert system could be used to select a unit type that suits the sub-problem best.

6.3.2 AI and Neural Networks working together

There is a variety of reasons for combining AI and neural networks in integrated systems. For example, there is strong support to believe that in humans the raw data taken from the environment is used to gain experience which leads to forming logical definitions and concepts. These concepts are manipulated logically using symbols, but the symbols are grounded on sensory input. Symbols are well suited for strong (or deep) logic where concepts are needed to represent insights that have to be communicated (for manipulation by other components or for explaining what is being done). Traditional AI focuses on representation and manipulation of symbols and is thus well suited to perform these tasks. However, one of the problems of AI is connecting it to the real world to get data into and out of the symbol system. There is a gap between real world data and symbol processing which is hard to bridge with traditional AI.

Neural networks are based on lower level representations and computations than traditional AI. They are good at processing large amounts of low level data, identifying and extracting relevant information. They also operate well in the presence of noise and imprecision. This makes neural networks suitable for bridging the gap between the real world and AI. It is possible to use neural networks for transforming masses of sensory input into symbols. Special purpose networks can be used for processing different sensory tasks like pattern recognition and identifying abnormal states. Neural networks can generate symbols for strong logic by building meaningful, experience based representations. They only represent what is really needed for a task, not what seems relevant. This makes their representations less arbitrary than those generated by knowledge engineers. Based on their ability for association and generalisation neural networks can also perform other tasks that can be learned by examples (e.g. learning words). However, many tasks are performed much more effectively if explicit rules can be used. For example understanding or generating sentences is much easier if a grammar (a set of rules) is available.

In general neural networks are good at performing low level tasks that need not be explained whereas traditional AI is good at high level tasks. The combination of the two fields covers an impressive range of tasks.

On a more practical level combining the two fields can be done in different ways. A neural network could be used for rule selection, i.e. to decide which rule to apply at a given state. An expert system could be used to determine which problem solving strategy to apply at a given state. A neural network can thus be used as a module (agent, knowledge source) in a hybrid system.

6.4 Summary

Applications of neural networks within AI can be useful. Neural networks have been used to implement AI systems like production systems. However, these systems have few advantages over traditional AI implementations and they do have disadvantages.

Neural networks are also used to solve problems that AI cannot solve convincingly or that are solved more elegantly with neural networks. The three main areas where neural networks have been applied to traditional AI problems are:

- understanding and generating speech and natural language,
- robotics (control of movement, environment representation),
- vision.

Examples of cooperative problem solving where AI techniques and connectionist techniques are used to supplement each other are:

- federative systems that can have knowledge based agents and neural network agents,
- integrated systems that use different paradigms for different purposes,
- refinement of knowledge bases using neural networks,
- using knowledge bases to incorporate *a priori* knowledge in neural networks,
- two systems performing the same task, the neural network to be used during normal operation, the rule based system to be used in abnormal situations to ensure safety.

There is need for both symbolic and sub-symbolic processing. Traditional AI and neural networks can be used to solve different parts of complex problems.

7 Conclusion

A summary of the strengths and weaknesses draws a picture of neural networks' potential.

The main weaknesses are:

- lack of theoretical foundations,
- difficult design: find a suitable network structure, sufficient input information, and appropriate training with representative training data,
- obscurity: they are based on and operate with terms that humans find hard to interpret,
- (intermediate) results cannot be explained or justified,
- networks scale up badly,

although some of these are being overcome by new developments.

The main strengths are:

- their generality: uniform methods for a variety of tasks,
- their simplicity,
- their adaptivity: learn and possibly forget and thus adapt to changing requirements,
- their ability for correlation detection: association, classification,
- their ability for feature selection: find most relevant features, improve discrimination, generalisation,
- their tolerance against damage: processing performance degrades gracefully,
- their tolerance against missing information: results degrade gracefully,
- their tolerance against noisy data in training and operation,
- their pragmatic approach: find a good solution fast, not the best solution slow or never,
- their fuzzy matching ability which allows a margin of error,
- their suitability to deal with uncertainty reflecting belief,
- their ability for self programming: find problem solving process,

- their alternative view: make it easier to think about a problem,
- their connection to physiology.

The field of neural networks is still developing. The exact limitations of its more complex techniques are still unknown. However, known limitations are being overcome. Theoretical foundations are being established and issues like modularity and specialisation are being investigated. In spite of their problems there are already a number of applications where neural networks have impressively outperformed traditional techniques providing an alternative approach. This approach suits some application areas particularly well.

In comparison with traditional approaches neural networks are superior in their abilities to generalise, guess and deal with noise. These functions are supported by the basic abilities of neural networks: association and similarity operations. Neural networks are also adaptive systems because they are learning systems.

In areas like strong logic and storage and retrieval of data they do worse than traditional systems. This is due to the fact that neural networks are bad on accuracy and they will make mistakes, but they are also tolerant towards external mistakes: they can deal with incorrect or imprecise information.

Neural networks operate on a sub-symbolic level. They are good at dealing with the kind of information quality that is found in the real world. In traditional AI this is one of the main problems: there is a gap between the symbol processing of AI systems and real world data. Neural networks can overcome this gap by generating symbols from low level data taken from the real world. Traditional AI and neural networks can be combined in cooperative systems where each part of the overall task is performed by the most appropriate technique: reasoning problems that require strong logic are best performed by rule based systems whereas tasks like signal processing, pattern recognition, learning from experience and motor control can be performed by neural networks.

References

[AR88] James A. Anderson and Edward Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, 1988.

[BL91] W. R. Becraft and P. L. Lee. Using Neural Networks for Diagnostic Focus in Chemical Process Plants. submitted to IEA/AIE-91, 1991.

[BLN91] Warren B. Becraft, Peter L. Lee, and Robjer B. Newell. Integration of Neural Networks and Expert Systems for Process Fault Diagnosis. In *Proceedings of the International Joint Conference on AI*, pages 832–837. Morgan Kaufmann, 1991.

[Byt87] Byte, October 1987.

[DSS92] J. Dunker, A. Scherer, and G. Schlageter. Integrating Neural Networks into a Distributed Knowledge Based System. In *12th International Conference on AI, Expert Systems and Natural Language, Avignon, France*, 1992.

[El91] Willem J. M. Epping and Albert R. l'Istelle. A Neural Data Compression Procedure for Seismic Images. In *Proceedings of the European Conference on Artificial Intelligence in Petroleum Exploration and Production*, Rueil-Malmaison, France, October 1991. Shell Research B.V, Institut Francais du Petrole.

[Fah88] Scott E. Fahlmann. An Empirical Study of Learning Speed in Back-Propagation Networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, September 1988.

[Fah91] Scott E. Fahlmann. The Recurrent Cascade-Correlation Architecture. Technical Report CMU-CS-91-100, Carnegie Mellon University, May 1991.

[FL90] Scott E. Fahlmann and Christian Lebiere. The Cascade-Correlation Learning Architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[Fu90] Li-Min Fu. Building Expert Systems on Neural Architecture. In *IEE Conference on Artificial Neural Networks*, pages 221–225, 1990?

[Heb49] D. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[Hil91] David Hillman. Knowledge Systems Based on Cascading Neural Nets. *AI Expert*, 6(12):46–53, Dec 1991.

[Hin89] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[HS87] R. Hartley and H. Szu. A comparison of the computational power of neural network models. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume 3, pages 17–22, 1987.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[Jac88] R. Jacobs. Increased Rates of Convergence Through Learning Rate Adaption. *Neural Networks*, 1, 1988.

[Khe92] Sukhdev Khebbal. An Object-Oriented Environment For Heterogenous Intelligent Systems. Technical report, Department of Computer Science, UCL, London, 1992.

[Koh82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Koh88] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, 2 edition, 1988.

[LeC89] Y. LeCun *et.al.* Backpropagation Applied to Hand Written Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[LLTT91] Joo-Hwee Lim, Ho-Chung Lui, A. H. Tan, and H. H. Teh. INSIDE: A Connectionist Case-based Diagnostic Expert System Thet Learns Incrementally. In *Proceedings of IJCNN-91*, pages 18–21, Singapore, November 1991.

[LLW92] Joo-Hwee Lim, Ho-Chung Lui, and Pei-Zhuang Wang. A Framework for Integrating Fault Diagnosis and Incremental Knowledge Acquisition in Connectionist Expert Systems. In *Proceedings of the Thenth National Conference on Artificial Intelligence*, pages 159–164. AAAI, AAAI Press / The MIT Press, July 1992.

[Min86] Marvin Minsky. *The Society of Mind*. Simon and Schuster, New York, 1986.

[MP43] Warren S. McCulloch and Walter Pitts'. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP88] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, MA, 3 edition, 1988.

[Pin90] Gadi Pinkas. Representing Propositional Logic in Interactive Symmetric Networks. In *presented at the MIND Conference*, Dallas, October 1990.

[Pom89a] D. A. Pomerlau. ALVINN: An Autonomous Land Vehicle In a Neural Network. In D.S.Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan Kaufmann, 1989.

[Pom89b] D. A. Pomerlau. ALVINN: An autonoomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Carnegie Mellon University, January 1989.

[QS88] N. Qian and T. J. Sejnowski. Predicting the Secondary Structure of Globular Proteins Using Neural Network Models. *Journal of Molecular Biology*, 202:856–884, 1988.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1-3. MIT Press, 1986.

[SL89] Patrick M. Shea and Vincent Lin. Detection of Explosives in Checked Airline Baggage using an Artificial Neural System. In *Proceedings of IJCNN-89*, volume II, pages 31–34, Washington, 1989.

[SR86] T. J. Sejnowski and C. R. Rosenberg. NETtalk: a parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, Johns Hopkins University, Electrical Engineering and Computer Science, 1986.

[SS91] H. T. Siegelmann and E. D. Sonntag. On the Computational Power of Neural Nets. Technical Report SYCON-91-11, SYCON, November 1991.

[SS92] William E. Staib and Robert B. Staib. The Intelligent Arc Furnace Controller: A Neural Network Electrode Position Optimization System for the Electric Arc Furnace. In *to be presented at IJCNN-92, Baltimore*, June 1992.

[TCS91] G. G. Towell, M. W. Craven, and J. W. Shavlik. Constructive Induction in Knowledge-Based Neural Networks. In G. Collins L. Birnbaum, editor, *Proceedings of the 8th International Workshop*, San Mateo, Ca., 1991. Morgan Kaufmann.

[TH86] D. S. Touretzky and G. E. Hinton. A Distributed Connectionist Production System. Technical Report CMU-CS-86-172, Carnegie-Mellon University, 1986.

[TSN90] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the 8th National Conference on AI, AAAI-90*, pages 861–866, 1990.

[Was89] P. D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold, New York, 1989.

[Whi88] H. White. Economic prediction using neural networks: the case of IBM daily stock returns. In *Proceedings of the IEE International Conference on Neural Networks,*, volume 2, pages 451–459, 1988.

[WS90] Paul Wilk and Jussi Stader. Applications of Neural Networks. In Terri Lydiard, editor, *Proceedings of the Workshop on Novel AI Techniques for Building Engineering Systems*, Edinburgh, December 1990. Artificial Intelligence Applications Institute, Report AIAI-IR-8.