Feedforward Neural Networks with Constrained Weights

Altaf Hamid Khan

A thesis submitted in satisfaction of the requirements for
the degree of Doctor of Philosophy

University of Warwick
Department of Engineering

August 1996

Contents

List of Figures

List of Tables

Abbreviations and Symbols

BP	Error backpropagation learning algorithm
CWN	Continuous-weight network
DNF	Disjunctive normal form
IWN	Multilayer feedforward network with integer weights
MFN	Multiplier-free feedforward network
MIMO	Multiple-input multiple-output
RMS	Root mean squared
SIMO	Single-input multiple-output
SISO	Single-input single-output
XOR	The Exclusive OR function

$\rightarrow$	approaches or converges to
$\uparrow$	$a \uparrow b$ means $a < b$ and $a \rightarrow b$
$\backslash$	The set $A \backslash B$ consists of elements of set A which are not elements of B
$\in$	belongs to
$\subset$	is a subset of
d:q:o	Feedforward network with d inputs, q hidden neurons, and o outputs
$\overrightarrow{d:q:o}$	d:q:o network with skip-layer synapses
$(\cdot)_{old}$	Value of '$\cdot$' calculated in the previous step
α	Empirical constant in the integer-weight learning procedure
α	An irrational number

α_χ	Weight discretisation coefficient
α_ρ	Black-hole radius coefficient
$\mathcal{B}(w)$	Black-hole function
b_w	Weight depth
β	Empirical constant in the integer-weight learning procedure
β_χ	Weight discretisation coefficient
β_ρ	Black-hole radius coefficient
χ	Rate of weight discretisation
C_R	Reinitialisation epochs
C_T	Training epochs
$C(\cdot)$	Set of all functions on the domain '$\cdot$'
Δw_{old}	The previous weight modification
δw	Smallest modification that can be made to a weight
δw	Perturbation in weight
d	Dimension of the input vector
ε	Maximum acceptable error in the network output
E_o	Error in the output of the network
E_{o_m}	Error of the output neuron with the maximum error during an epoch
$E_{o_{rms}}$	Average RMS error in the network output for one epoch
E_w	Weight discretisation error
η	Learning rate
$f(\cdot)$	Target function to be approximated by the network
$\hat{f}(\cdot)$	Network output
μ	Momentum
$\mathbb{N}$	The set of all natural numbers, i.e. all integers > 0
$\mathcal{N}(\cdot)$	Feedforward network
o_j	Response of the jth output layer neuron
$\psi(\cdot)$	Neuron activation function
q	Number of hidden neurons
$\mathbb{Q}$	The set of all rational numbers

$\mathcal{Q}(w)$	Weight discretisation function
ρ	Black hole radius
$\mathbb{R}$	The set of all real numbers
$\mathbb{R}^d$	The set of all all points in the real d-dimensional space
RND	Random number selected uniformly from $(0, \pi/2)$
$\text{sgn}(\cdot)$	Signum function, provides the sign of the argument or 0 if argument is 0
$\sigma(\cdot)$	Neuron activation function
$\sin(\cdot)$	Sine function
$\sup\{\cdot\}$	Supremum of $\{\cdot\}$
t_j	Target or desired value for the response of the jth output layer neuron
$\tan(\cdot)$	Tangent function
$\tanh(\cdot)$	Hyperbolic tangent function
θ_j	Offset of the jth neuron
u_j	Output of the jth neuron
w_j^o	Synapse connecting the jth hidden neuron to the output neuron
$\mathbf{w}^o$	Output layer weight vector
w_{ij}	Synapse connecting the ith input to the jth hidden neuron
$\mathbf{W}$	Weight vector of a network
W	Total number of weights in a network
x	Network input
$\mathbf{x}$	Network input vector
$\mathbb{Z}$	The set of all integers

Acknowledgements

The author would like to thank the the following persons and organisations:

Dr. Roland Wilson and Prof. David Whitehouse, for taking him on as their student at such a late stage in his studies. For stimulating conversations, productive guidance, and constant encouragement.

Mark Craven, Andy Larkin, Andy Pardoe, and Nick Porter for maintaining an enjoyable working environment during occasionally difficult but often interesting times. Especially Nick Porter, for his generous assistance in all matters computable.

Commonwealth Scholarship Commission in the United Kingdom for their financial support.

University of Engineering and Technology, Lahore, Pakistan, for granting him leave for postgraduate study.

Declaration

All work reported in this thesis was carried out by the author unless stated otherwise. All of author's work reported in this thesis was carried out during his period of study for the PhD degree. The publications which have resulted from the author's work on this thesis appear in Appendix D.

Summary

The conventional multilayer feedforward network having continuous-weights is expensive to implement in digital hardware. Two new types of networks are proposed which lend themselves to cost-effective implementations in hardware and have a fast forward-pass capability. These two differ from the conventional model in having extra constraints on their weights: the first allows its weights to take integer values in the range $[-3, 3]$ only, whereas the second restricts its synapses to the set $\{-1, 0, 1\}$ while allowing unrestricted offsets. The benefits of the first configuration are in having weights which are only 3-bits deep and a multiplication operation requiring a maximum of one shift, one add, and one sign-change instruction. The advantages of the second are in having 1-bit synapses and a multiplication operation which consists of a single sign-change instruction.

The procedure proposed for training these networks starts like the conventional error backpropagation procedure, but becomes more and more discretised in its behaviour as the network gets closer to an error minimum. Mainly based on steepest descent, it also has a perturbation mechanism to avoid getting trapped in local minima, and a novel mechanism for rounding off 'near integers'. It incorporates weight elimination implicitly, which simplifies the choice of the start-up network configuration for training.

It is shown that the integer-weight network, although lacking the universal approximation capability, can implement learning tasks, especially classification tasks, to acceptable accuracies. A new theoretical result is presented which shows that the multiplier-free network is a universal approximator over the space of continuous functions of one variable. In light of experimental results it is conjectured that the same is true for functions of many variables.

Decision and error surfaces are used to explore the discrete-weight approximation of continuous-weight networks using discretisation schemes other than integer weights. The results suggest that provided a suitable discretisation interval is chosen, a discrete-weight network can be found which performs as well as a continuous-weight networks, but that it may require more hidden neurons than its conventional counterpart.

Experiments are performed to compare the generalisation performances of the new networks with that of the conventional one using three very different benchmarks: the MONK's benchmark, a set of artificial tasks designed to compare the capabilities of learning algorithms, the 'onset of diabetes mellitus' prediction data set, a realistic set with very noisy attributes, and finally the handwritten numeral recognition database, a realistic but very structured data set. The results indicate that the new networks, despite having strong constraints on their weights, have generalisation performances similar to that of their conventional counterparts.

1

Feedforward Networks

1.1 Implementing Feedforward Networks in Hardware

Ease of hardware implementation is one of the features which distinguishes the feedforward network from competing statistical and machine learning techniques. The most distinctive feature of the graph of a feedforward network is its homogeneous modularity. Because of its modular architecture, the natural implementation of this network is a parallel one, whether in software or in hardware. Although most current activity is in software implementations on serial computers [127], the unique benefit of the feedforward network, i.e. fast speed of execution, can only be achieved through its realisation in parallel hardware: electronic or optical, analogue or digital. Of these parallel realisations, the digital electronic is the one that holds the most interest currently – the modular architecture of the feedforward network is well matched with the current VLSI design tools and therefore lends itself to cost-effective mass production. There is, however, a hitch which makes this union between the feedforward network and digital electronic hardware far from ideal: the network parameters (weights) and its internal functions (dot products and activation functions) are inherently analogue. It is too much to expect a network trained in an analogue (or high resolution digital)

environment to behave satisfactorily when transplanted into a low resolution hardware setup. Use of the digital approximation of a continuous activation function, and/or range-limiting of weights should, in general, lead to an unsatisfactory approximation due to the nonlinear nature of the network. The solution to this problem may lie in a bottom-up approach – instead of trying to fit an analogue network in hardware after training, train the analogue network in such a way that it is suitable for implementation in hardware after training. This approach is the main theme of this thesis.

The main purpose of this chapter is to present the principal concepts behind the conventional multilayer feedforward network and to define some of the terminology that will be invoked in the coming chapters. Many of the concepts presented here will be discussed in more detail in later chapters.

After explaining the structure and functionality of the prototype feedforward network, some of its recent applications will be presented. Its connections with biological and statistical models will then be briefly discussed. The answers to the questions 'what is this network capable of?' and 'how difficult is it to train?' are the topic of the next two sections. Pre-training data processing techniques and an overview of methods for network training are then presented. The ability of a network to learn the main concepts hidden in noisy data will be discussed. Some examples of its implementation in hardware are also presented. In view of the hardware implementation requirements, the capabilities of networks with constraints placed on their weights are then explained. After a short note on the major milestones in the development of feedforward networks, this chapter concludes with an outline of this thesis.

1.2 Multilayer Feedforward Networks: Introduction

Multilayer feedforward networks are general-purpose modelling devices which can extract the functionality of the underlying process from examples generated by that process. Each of these examples consists of the input vector and the response of the process to that input vector. Feedforward networks are non-parametric in their modelling ability, in the sense that they do not demand any structural information about the process

that they are modelling, merely its characteristic in the form of a set of input/output examples. They "let the data speak for itself" [44].

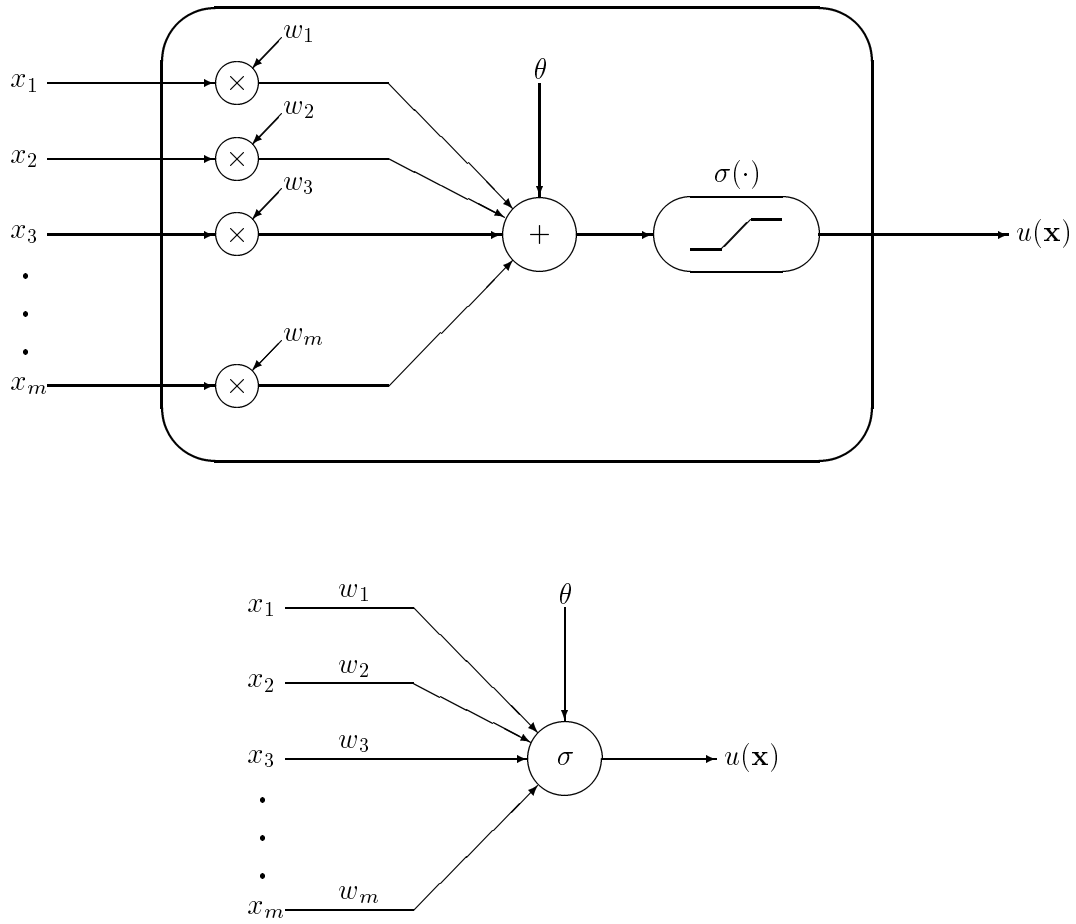
Figure 1.1 shows the details of the fundamental processing element used in forming a feedforward network. This element is conventionally known as the artificial **neuron**[1,2] because its function is somewhat analogous to the biological neuron. Each neuron calculates the dot product of the incoming signals $\mathbf{x}$ with its synaptic strengths $\mathbf{w}$, adds the offset θ to the resultant, and outputs a value which is calculated by applying an **activation function** to that sum. Two of the most common nonlinear activation functions, logistic and hyperbolic tangent, are shown in Figure 1.2.

Figure 1.3 shows how the artificial neurons can be connected to form the conventional multilayer feedforward network. From now onwards, this network will be referred to as the Continuous-Weight Network (CWN). This d:q:1 CWN has only a single output, but in general it can have any number of outputs. For analytical studies however, one needs to study only this single output prototype, as any k-output CWN can be represented as an ensemble of k separate, single output CWNs. This particular CWN, having two layers of neurons, is termed a **2-layer** CWN. The input layer is not counted as a layer as it does not perform any processing and consists of fan-out units only. The middle layer is termed the **hidden layer**, due to its lack of direct connections with the outside environment. The edges connecting the nodes to each other are called **synapses**. The extra vertical edges coming into the hidden nodes are termed **offsets**.[3] It is assumed that the unconnected end of the offset edges is connected to a constant value of 1. The term **weight** is used to refer both to the synapses and offsets. The neurons in the hidden layer must have nonlinear activation functions for the CWN to be able to perform nonlinear mappings. If the hidden neuron activation functions are linear, the 2-layer CWN can be collapsed into a 1-layer d:1 CWN, commonly known as

[1] Important new terms appear in **boldface** on first usage and their definitions are included in the glossary as well.

[2] Also known as a node.

[3] An offset for the output neuron is not required for the 'universal approximation' property (to be discussed in Section 1.6) of the CWN to hold, but has customarily been used by the practitioners.



$$u(\mathbf{x}) = \sigma(\mathbf{x} \cdot \mathbf{w} + \theta) = \sigma\left(\sum_{n=1}^{m} w_n x_n + \theta\right)$$

Figure 1.1 Detailed (top) and symbolic (bottom) representations of the artificial neuron.

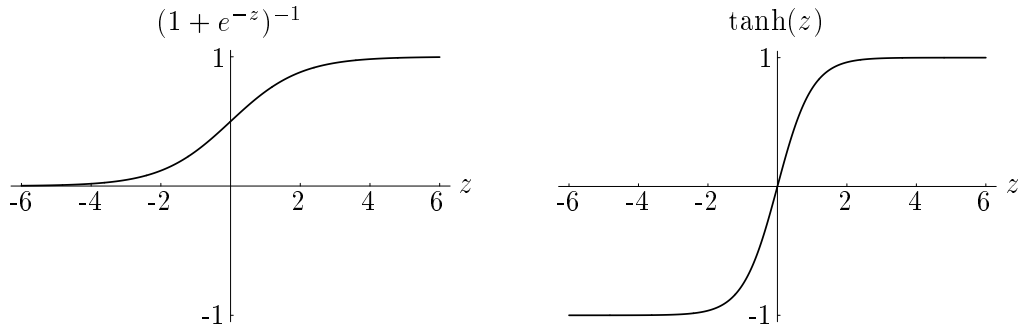
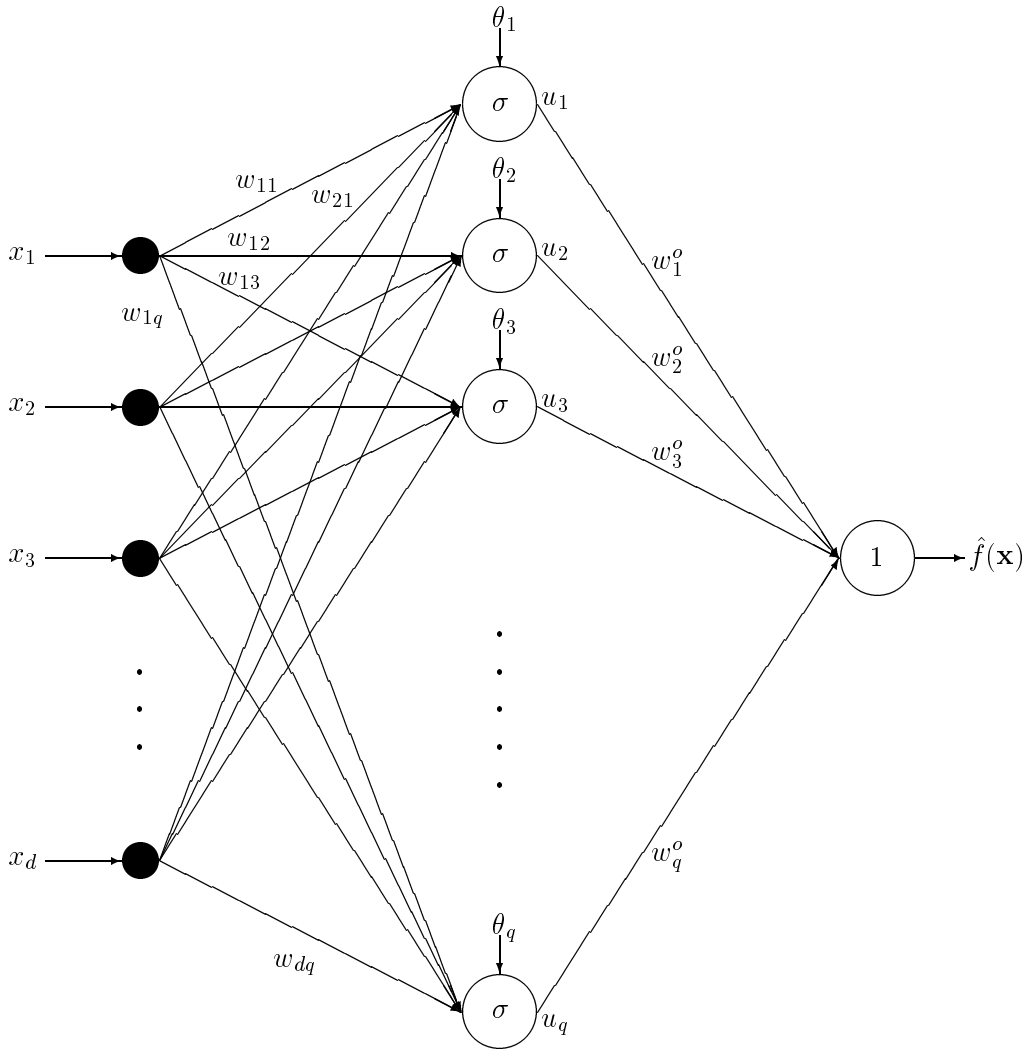


Figure 1.2 Examples of activation functions, $\sigma(\cdot)$: logistic and hyperbolic tangent.



$$u_j = \sigma(\mathbf{x} \cdot \mathbf{w}_j + \theta_j), \qquad \hat{f}(\mathbf{x}) = \mathbf{u} \cdot \mathbf{w}^o = \sum_{j=1}^{q} w_j^o \, \sigma\left(\sum_{i=1}^{d} w_{nj} x_n + \theta_j \right).$$

Figure 1.3 The conventional 2-layer d:q:1 feedforward network with d inputs, q hidden neurons, and a single output.

a **perceptron**.[4]

This thesis discusses the 2-layer CWN, from now on termed as CWN only, exclusively. CWNs with many hidden layers may have advantages in having more profound mapping capabilities – they can implement a mapping with fewer weights as compared with a 2-layer network having a similar performance. They are not, however, as well understood as 2-layer networks and are harder to train [3].

1.3 Application Examples

It has been 10 years since the pioneering work of Rumelhart et al. [123], in which learning in multilayer feedforward networks was first introduced. A large number of papers on the application of CWNs have since been published, in areas of application ranging from medicine, finance, process industry, high energy physics, automotives, telecommunications, robotics, to aerospace [85,111,159]. These applications are generally divided into two groups: **function approximation**[5] and **classification**. This division is based on the type of the desired outputs required to accomplish the task. If the output values are continuous, the CWN is performing function approximation, whereas if the outputs are restricted to a finite set of values, it is doing classification. The examples presented below highlight recent applications in high energy physics, automotives, meat industry, and nuclear material processing.

A fully parallel hardware implementation of a feedforward network is being used for the closed-loop real-time control of the shape of the magnetic confinement field of a high-temperature plasma in a Tokamak fusion reactor [17]. This feedforward network, having digitally stored weights but analogue signal paths, and a bandwidth of ≥ 20 kHz, was trained using analytically generated data to simultaneously control the currents in

[4] Only an identity activation function is required in the output neuron for the 'universal approximation' property (to be discussed in Section 1.6) of the CWN to hold. It is however customary to use a logistic or hyperbolic tangent function for 'classification tasks' (to be defined in the next section). Having a logistic function as the activation function in output neurons also helps in the probabilistic interpretation of their responses [12].

[5] Also known as regression.

all of the coils generating the confinement magnetic field.

An example of an application in automotive safety is Delco Electronics' supplemental inflatable restraint (air bag) controller which uses a feedforward network to distinguish between deployment and nondeployment events [70]. Time series data collected from instrumented vehicle crash tests, along with the requirements of when and if air bag must be inflated for different type of situations, was used to train the feedforward network.

The Danish Meat Research Institute has recently announced[6] that it has successfully completed the trial run on the prototype meat classifiers and will be deploying 8-10 units of these completely automated systems during the next year. These classifiers determine the value of cow carcasses, with the help of weight and visual information. It employs three feedforward networks to extract the class (cow, bull, or heifer) and shape of the carcasses which are then used in a linear model to determine the payment to be made to the farmer [19].

Urneco (Capenhurst) Ltd. employs a pair of feedforward networks for the closed-loop control of copper lasers, which are used in isotope separation for uranium enrichment [24]. These feedforward networks have been trained according to the actions of experienced human operators to avoid certain load discharge conditions that reduce the lifetime of expensive laser modulator components and therefore improve the efficiency, due to decreased running costs and down time.

1.4 Biological -vs- Artificial Neural Networks

Although the original inspiration for work in the artificial neural network area was the human brain, the emphasis has since shifted from biological plausibility to usefulness as a computational tool. Artificial multilayer feedforward networks do, however, share some features with the biological brain [38]: both are layered structures formed by a number of homogeneous and simple processing elements – neurons. Both types of neurons have many inputs and produce an output signal which is a nonlinear function

[6] In a press conference on 25 June 1996.

of the dot product of inputs and weights. The key factor that distinguishes neural networks, both biological and artificial, from traditional computing paradigms is that processing is asynchronous and local to the individual neurons. The major emergent properties that these two classes of networks have in common are associative memory and a degree of fault tolerance [38].

The main barrier to the wide acceptance of the plausibility of artificial feedforward networks as analogues of the real ones is the learning procedures. It is well known that synapses in a human brain change with experience, but the exact mechanisms are not well understood because of the complex and distributed nature of the system. There is some agreement, however, that it is nothing like the popular procedures used for training feedforward networks. This, due to the fact that those procedures involve some global computation steps, which is in violation of the strictly local theories of learning in biological networks [39].

1.5 The Statistical Connection

Many neural network paradigms have their analogues in the statistical arena. The two fields, however, cannot be considered as being identical twins – more like step-brothers. The main differences between them are perhaps more cultural than technical [107]: statistics has its roots in mathematics, neural networks in engineering, biology, and computer science. Statisticians are usually conservative and pessimistic, neural folk enthusiastic. Statisticians are often concerned with small samples, neural workers with large data sets. Statisticians mostly deal with static data, while neural workers are also interested in **in-situ learning**[7]. The majority of statisticians do mathematical modelling, neural workers run computer solutions. Researchers in the artificial neural network field also differ in having a long-term goal of designing artificially intelligent systems. Both communities can, however, benefit from each other's expertise: for example, only recently the neural network community has started to benefit from the

[7] In-situ learning is performed by a deployed network which has already been trained. The deployed network constantly adapts its weights with respect to changes in the input/output behaviour of its target task.

rigorous frameworks, such as Bayesian techniques, available in statistics [11, 113].

The feedforward network, the only neural paradigm discussed in this thesis, has a direct analogue in statistics: **projection pursuit regression** [66]. Projection pursuit is a generalisation of the CWN in that it allows more than one type of activation function in the hidden layer. These non-homogeneous activation functions are data-dependent and constructed during learning. Projection pursuit learning differs from conventional CWN learning in that it is performed one hidden neuron at a time. The output weight of a hidden neuron is optimised followed by the shape of the activation function and then the input weights. It will be discussed in the next section[8] that the CWN can approximate almost all functions[9] to any desired accuracy. Projection pursuit regression, a generalisation of CWN, must also be able to approximate almost all functions, and therefore does not hold any advantage in that area. The advantage may, however, lie in its ability to construct more compact representations of arbitrary training data sets due to its freedom in having data-dependent activation functions.

1.6 Approximation Properties

2-layer CWNs are universal approximators in the space of **Borel measurable functions**[10] (see for example [63]). In other words, a 2-layer CWN exists that can, given enough training data and enough hidden neurons, approximate virtually any function of interest to any desired degree of accuracy [127]. This is a very powerful statement and provides great comfort to experimentalists in reinforcing their beliefs about the capabilities of CWNs. This, however, guarantees only the *existence* of an approximating network and does not give any clues about how to construct one.

Correspondingly, loading a CWN, that is finding the set of optimal weights for a CWN for a given set of training data, is an intractable problem. This problem has been shown to belong to a class of very tough problems, called **NP-complete** problems,

[8] Also in Chapter 2.

[9] See next section for the definition of 'almost all'.

[10] Just about all functions that one may encounter are Borel measurable. Functions that are not Borel measurable do exist but are known to mathematicians only as mathematical peculiarities.

which are generally accepted to have no polynomial-time solution [42]. The difficulty of solving problems belonging to this class increases exponentially with the number of inputs. Intractability of loading, however, is the case for a fixed-size CWN only – if hidden neurons can be added and eliminated during training then loading is tractable. Moreover, fixed-size CWNs can be trained to achieve 'good enough' solutions instead of optimal ones, which, in practice, is not as time consuming as achieving optimal solutions, and is certainly better than having no solution at all [42].

1.7 Feedforward Network Training: Prerequisites

The cardinal choice in the training of a CWN is the choice of the number of neurons in the hidden layer. This number is a function of the complexity of the concept to be learned from the training data. In most cases, the complexity of that concept is not known prior to training. In those cases, one of the following three schemes can be employed: training several networks, ontogenic techniques, reducing the complexity of a large network. The most frequently used technique is to start by training the simplest network and try several networks with an increasing number of hidden neurons until the required performance is achieved. Ontogenic schemes are self-constructing techniques, of which the **Cascade-correlation** method is the most well known example [37]. This method starts with a network without any hidden neurons and systematically increases their number during training until the required performance is achieved. Finally, one can also start with a large network and try to decrease its complexity during training to match the complexity of the concept being learned. The thinking behind this technique can be explained by the analogy of fitting an arbitrarily shaped object in a container which is only slightly larger than the object in all dimensions. In general, fitting that object in the container will be quite difficult, if possible at all. It is simpler to start with a large container, place the object in the container, and then somehow shrink[11] the size of the container to achieve a very snug fit.

Pre-training data processing can be used to reduce training time and to improve

[11] Such 'shrinking' techniques in regard to CWNs will be discussed in Section 6.3.

the quality of learning. For example, the standardisation of inputs to zero-means and small magnitudes results in all the input vectors clustering around the origin. This, combined with the initialisation of the CWNs at the start of training with small randomly distributed weights which results in the start-up decision boundaries bunching up around the origin, gives the CWN the best chance of fast learning [129]. Moreover, Le Cun et al. have shown analytically that standardising the inputs to zero mean improves the convergence properties of some learning procedures [82].

For discrete inputs or outputs, categorical variables should not be treated as continuous variables [13]. For instance, one of the input variables for the training data to be used in Section 6.5.2 represents shapes and can have one of three values: round, square, or octagon. This variable will not be represented as a single tri-state input, as that imposes an ordering of square being somehow greater than round and less than octagon [13], but as a triplet, having three possible states $(1,0,0), (0,1,0)$, and $(0,0,1)$ depending upon the shape being round, square, or octagon, respectively [144].

In classification problems, outputs are generally encoded as $\{0,1\}$ or $\{-1,1\}$. Due to the fact that weight modifications computed by some learning procedures are proportional to the output value of a neuron [122], the first encoding has the disadvantage in that the weight modifications calculated by the learning procedure for class 0 are smaller relative to the ones for class 1. The bipolar binary encoding does not have this problem but suffers from the drawback that no learning takes place for the most 'confused' neurons, i.e. the ones with output values around zero. On balance, the bipolar-binary encoding should be preferred as in that case the learning procedure treats both output states in the same fashion during training.

1.8 Feedforward Network Training: Procedures

A training procedure is a collection of heuristics that finds appropriate weights for a particular CWN, given a set of training examples, such that an appropriate **cost function**, also known as an error measure, is minimised. The most often used error

measures, $\boldsymbol{L_p}$-norm measures, have the general form of:

$$E_o(\mathbf{W}) = \left(\sum_{j=1}^{J} |t_j - o_j|^p \right)^{\frac{1}{p}}, \qquad p \in \mathbb{N}, \tag{1.1}$$

where $\mathbf{W}$ is the weight vector of the network, $\boldsymbol{t_j}$ is the target or desired value of the jth output, and $\boldsymbol{o_j}$ is its value computed by the CWN. Popular values for p are 1, 2, and ∞, with the $p = 2$ case, i.e. the least-squares measure, which emphasises the larger errors, being the most common one. The $p = 1$ and $p = \infty$ cases are mostly used in hardware implementations because they are easier to compute [59].

CWN training procedures can be divide into two major categories: those based on steepest descent methods and those influenced by stochastic optimisation techniques.

Steepest descent methods attempt to minimise the cost function by taking small steps on the **error surface**[12] in the direction of the maximum gradient. This process is analogous to a near-sighted skier trying to find the quickest way to the base of a mountain by skiing down the slopes with the largest gradient. A computationally efficient way of implementing this process in a CWN, called **error backpropagation** (BP), was popularised by Rumelhart et al. [122]. In the **batch**[13] version of BP, the weights are updated at the end of every **epoch**,[14] whereas in the **on-line**[15,16] version weights are updated after the presentation of every training example. Both versions are guaranteed to converge to a solution in appropriate circumstances (see Section 2.3). The latter version is, however, faster for large training sets having some degree of information redundancy among examples [14]. Two of the most common parameters related to BP are **learning rate**, $\boldsymbol{\eta}$, and **momentum**, $\boldsymbol{\mu}$: η determines the size of

[12] Error surface is the plot of the cost function with respect to all of the weights in a network.

[13] Also known as off-line and total gradient-descent learning.

[14] An epoch is the cycle in which all examples in the training set are presented to the network.

[15] Also known as instantaneous, incremental, pattern, and stochastic gradient descent learning.

[16] On-line learning differs from the in-situ learning mentioned in Section 1.5 in that the latter is the property of a *network* requiring the deployed network to have adaptive weights, whereas the former is a property of the *learning procedure*, requiring the weights to be updated on the presentation of every example.

the weight modification at each training step, and μ controls the effect of the weight modification of the previous step over the weight modification of the current step. If η is small and μ close to 1, on-line BP approximates batch BP [128]. As training examples are usually presented to a network in a random order, on-line BP does its search in the weight space in a stochastic manner, and therefore is less prone (compared with batch BP) to getting stuck in **local minima**[17] of the error surface [53]. **Weight perturbation** is an alternative to BP learning. In this method, all of the weights are perturbed in turn and the associated change in the output of the network is used to approximate local gradients [59]. This technique requires only feedforward calculations for its operation, which simplifies its implementation in hardware [166]. It lacks the mathematical efficiency of BP however, and therefore requires a large number of epochs to reach acceptable solutions.

Stochastic methods are the less common alternative to steepest descent. A popular representative of these methods is **simulated annealing**. This method is analogous to the physical process of annealing [69]. It is different from methods based upon steepest descent in that the myopic skier of the last paragraph is not always going downhill, but going downhill only 'most of the time' [54]. This way the skier will not get trapped in local minima. In this method, weight modifications are made permanent if the new value of the cost function is lower or equal to the old one. If the new value is higher then the weight change is accepted with a probability which diminishes with the number of epochs. Besides the global minimum search advantage, stochastic techniques have the added advantage that they do not require gradient computations, which is attractive from the hardware implementation point of view. The main drawback of this class of techniques is speed: they generally require a large number of epochs for convergence and each epoch generally requires the recalculation of the cost function for every training example and every weight change.

Steepest descent methods are fast but can get trapped in local minima, whereas stochastic techniques have the ability of finding global minima but are slow. Methods

[17] Local minima are points of zero gradient on an error surface which are not global minima. **Global minima** are the points of minimum error on an error surface.

that combine the speed of steepest descent with the global optimisation character of stochastic techniques are an attractive alternative. An example of this approach is a modified version of on-line BP with a diminishing learning rate: the modification being the addition of decreasing amounts of random noise to the weights at each weight BP modification step [75]. This procedure is guaranteed to converge to a global minimum.

1.9 Generalisation Performance

Generalisation performance[18] – the accuracy of a trained network on a set of data which is similar to but not the same as the training data set [53] – is the key metric which determines a learning paradigm's usefulness [15]. This metric can be maximised by selecting a data set which completely represents the concept to be learned, and using that set, along with a global-minimum finding procedure, to train a network having a complexity that matches that of the concept to be learned.

A trained network will be a good generaliser if it has learned the concept embedded in the training data [16]. Training a network to be a good generaliser is not a trivial task because most real-life applications demand training with noisy data. A good generaliser usually has a smooth input to output mapping, which generally means that it will not have many large weights [16]. More complex networks give a better fit to training data but are not good generalisers. The best generalisers are neither too complex nor too simple, but match the the complexity of the problem exactly. Training the best generaliser is quite difficult, just like fitting an object into a container which exactly matches the dimensions of the container. Poor generalisation results from the networks **over-** or **under-fitting** the training data. An over-fit is due to the network having a higher complexity than the concept embedded in the training data. This causes the network to essentially become a 'look-up table' for the training data [53]: the network behaves very well for the training data, but gives erroneous responses to inputs which are nearby but not actual training data. This situation in the one dimensional case is shown in shown in Figure 1.4(c). An under-fit is caused by the network having a

[18] Also known as statistical inference.

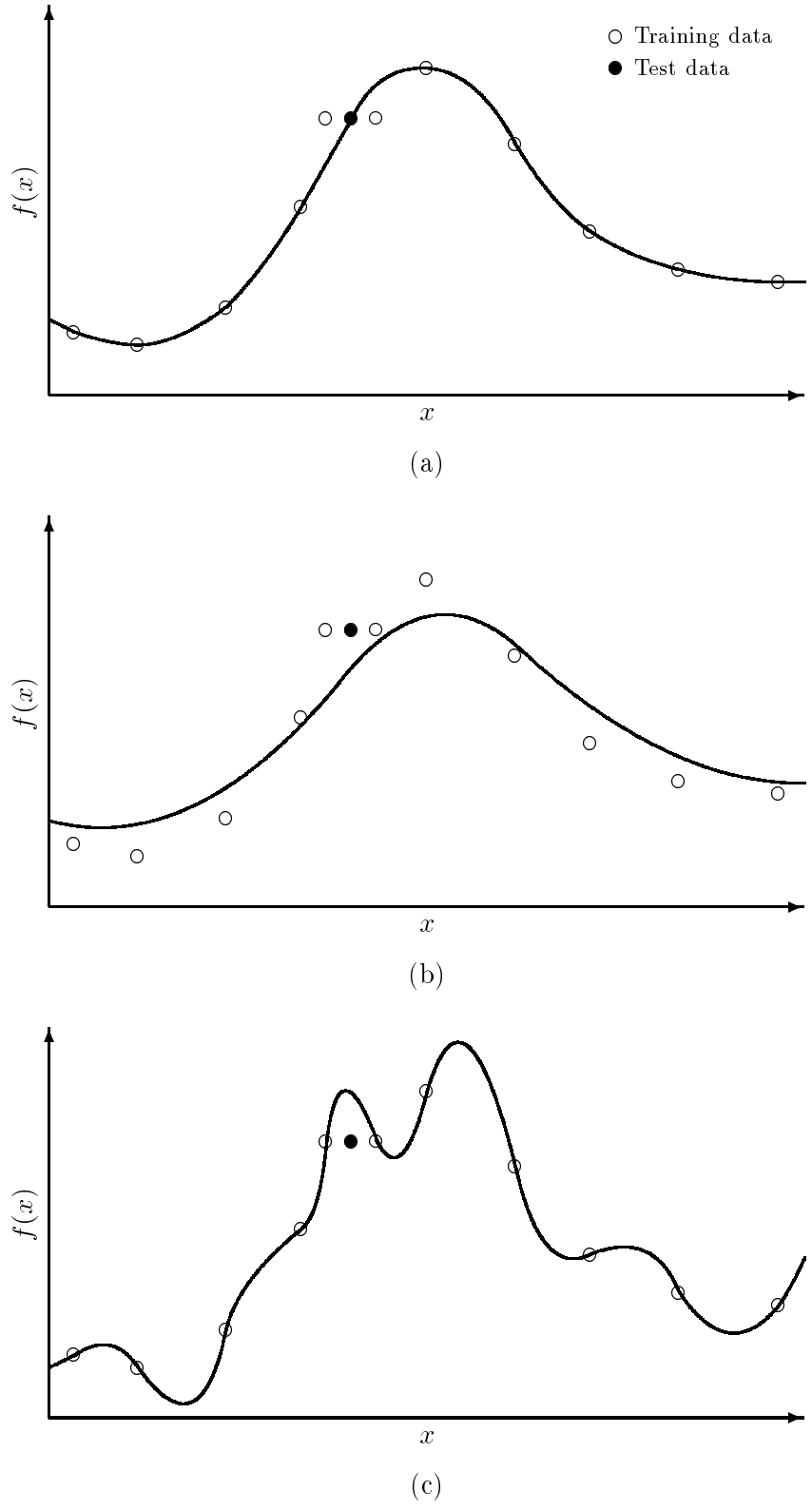


Figure 1.4 Example of a good fit (a), an under-fit (b), and an over-fit (c) of training data. The first case will results in good generalisation, whereas the latter two in poor.

complexity lower than that of the concept embedded in the training data. In general, it is harder to detect an under-fit during training as compared with an over-fit. The techniques for avoiding an over-fit and consequently improving generalisation performance, i.e. **regularisation techniques**, will be discussed in Chapter 6.

1.10 Hardware Implementation

Although most feedforward networks are implemented in software on serial computers [127], a very attractive feature of CWNs, i.e. fast speed of execution, can only be achieved through their realisation in parallel hardware: general purpose or customised, optical or electronic, analogue or digital, or any combination thereof. Custom hardware has the advantage in speed[19] and the disadvantage in cost. Electronic hardware is generally compact and cost-effective. Optical hardware has the advantage of free-space connectivity. Analogue electronic systems are usually very fast, but suffer from susceptibility to noise, manufacturing difficulties, and lack of non-volatile on-chip adjustable weights [56]. Digital electronic systems have the drawbacks of limited resolution computation and slow speed. They are less susceptible to noise and have non-volatile adjustable weights. Their main advantage is however in the ease of implementation due to the availability of a wide variety of mature VLSI tools and manufacturing facilities. This is the reason for their popularity, as is clear from the large number of reported systems [56, 67]. This section discusses digital electronic systems along with an analogue-digital electronic hybrid that has overcome the limited resolution computation drawback of the digital systems and the lack of non-volatile on-chip adjustable weights of analogue systems.

The *L Neuro 1.0* chip manufactured by Philips [92] is an example of a simple digital implementation of feedforward networks with adjustable weights. It can be configured as having 64 neurons with 8-bit weights or 256 neurons with 4-bit weights. Each neuron has 16 inputs. During learning, 16-bit weights must be used, allowing a maximum of

[19] And in many cases, better accuracy as well, because the hardware is designed to be very well matched with the target CWN.

32 neurons only. The output of the neurons are computed sequentially with the help of a serial-parallel multiplier which performs the products and accumulations for the matrix-vector products. The activation function calculation is done with the help of an off-chip look-up table – a major drawback. This chip was designed to do on-chip **Hebbian learning**[20] only, but can be trained with BP with the help of a host processor. More than one chip can be cascaded to increase the number of neurons or the number of inputs for each neuron.

The *RN-200* chip by Ricoh [108] is a unique design in that it implements its input, internal, and output signals as stochastic pulse trains. The advantage of this encoding is in the ease of computation: sums can be implemented as a logical OR, products as AND, and $1 - x$ as COMPLEMENT(x). The sigmoidal activation function is also implemented implicitly with an OR operation. Negative quantities are expressed in terms of inhibitory and excitatory synapses. There are 16 neurons on a chip, each having up to 16 inputs. Weights have a resolution of 8-bits. The computational accuracy of this chip depends upon the absence of correlation between pulse trains. This is achieved by implementing each weight with its own random number generator. The presence of on-chip BP training makes this chip very useful for embedded applications.

The *WSI*[21] neurocomputer from Hitachi [168] uses an input, an address, and an output bus to connect 1152 neurons in a SIMD[22] array. The address bus is used to sequentially select the 8-bit weights from common memory, and the dot product of inputs and weights is sequentially accumulated. A single activation function calculator is used by all neurons. The output bus is used to sequentially communicate the outputs to other neurons. The number of possible neurons is halved during on-chip BP training. For simplifying the implementation of BP, each neuron consists of two physical neurons during training. The resolution of weights is increased to 16-bit during training. An 8-bit $\boldsymbol{\Delta w_{old}}$ (the change in weight during the last modification step) is assigned to each

[20]The idea behind Hebbian learning is that the synapse between two neurons should be strengthened if they fire simultaneously.

[21]Wafer Scale Integration.

[22]Single instruction, multiple data

weight for the momentum term calculations. A unique feature of this implementation is that to reduce storage requirements, only the 64 largest weight values are ever kept in the memory – all others are forced to zero.

The analogue-digital hybrid feedforward network used in the Tokamak reactor of Section 1.3, which has an output resolution of at least 8 bits and a bandwidth of $\geq$ 20 kHz, was implemented as three separate VME-bus cards. The first card uses analogue electronics to buffer and normalise the inputs. The second one consists of 12-bit digital to analogue converters, configured as 4-quadrant multipliers of analogue signals with digitally stored 12-bit weights. The third card implements an analogue activation function, hyperbolic tangent, with the help of two temperature compensated transistors configured as a long-tailed pair. This hybrid scheme exploits the individual strengths of the analogue and digital domains – the non-volatile weight storage of the former and the high resolution and fast signal processing of the latter – while avoiding their weaknesses – the volatile weight storage of the latter and the low resolution and slow signal processing of the former.

At least one of two bottlenecks on processing speed and/or size of the implementation manifest themselves in all four of the above examples: storage of high-resolution weights and the multiplication operation. The multiplication operation is the main bottle neck due to the large number of multiplications required, inherent slowness of the serial multiplication operation, and the physical size of the multiplier. The main contribution of this thesis relates to the characteristics of the feedforward networks having low-resolution weights and the simplification or complete exclusion of the multiplication operation.

1.11 Feedforward Networks with Constrained Weights

With the exception of the last section, all the discussion so far has been about CWNs having continuous weights of arbitrarily large magnitudes. Such weights are not possible in practice. In digital electronic hardware, a designer has to determine the amount of memory required to store the weights by considering two factors [112]: the mini-

mum variation by which a weight can be modified, δw, and the difference between the maximum and minimum value of weights, the dynamic range. The ratio of these two factors determines the **weight depth** b_w, the number of bits of storage required for each weight:

$$b_w = \log_2 \frac{w_{max} - w_{min}}{\delta w}.$$

Clearly, any desired increase in either the precision or dynamic range of weights increases the weight depth. Weight depth not only affects the storage requirements but also the speed and cost of computational circuitry. Consider the multiplication operation: the number of multiplication operations required for a forward pass in a CWN is the number of synapses in that network. The quickest way to perform a multiplication is with a flash multiplier. An n-bit fixed-point VLSI flash multiplier consists of $n \times n$ full adders, each one of which is made up of 31 transistors. The slower but more cost-effective option is to have a single multiplier in each hidden and output neuron, and to perform the multiplications in a sequence instead of all in parallel. To reduce the transistor count even further at the expense of slower computations, a sequential multiplier can be used. Clearly, any desired increase in the weight depth results in slow and/or expensive multiplier.

The above discussion points towards placing constraints on weights due to the requirement of the smallest possible b_w for achieving fast and cost-effective hardware implementations. The question that now arises is: does this requirement effect the approximation capabilities of a feedforward net? If it does affect the approximation capability, then in what way? Is the reduction in capability gradual with b_w or abrupt? Can it somehow be compensated for by adjusting the only other variable in the CWN architecture: the number of hidden neurons? And what about the training procedure: are the conventional continuous weight methods appropriate for networks having weights with limited resolution? The answers to these questions will be discussed by considering three variations on the constrained weight theme: bounded weights, bounded weights having limited-resolution, and bounded and limited-resolution synapses, but continuous valued offsets.

Stinchcombe and White [137] have shown analytically that provided the activation

function meets certain criteria, the bounds on a CWN's weights can be traded off with a possible increase in the number of hidden neurons without sacrificing the universal approximation property. In particular, the input-layer weight vectors for the hidden neurons can be restricted to unit-magnitude provided that the activation function is a logistic or hyperbolic tangent function. Further details on this topic will be presented in Chapter 2. These networks can be trained using conventional techniques with the modification that disallows weights to exceed set bounds.

Networks with small discrete weights are ideal for digital implementation for having low storage requirements and a simplified multiplication operation. It will be shown in Chapter 3 that these networks do not possess the universal approximation capability. Experiments with these networks, in particular with networks having small integer weights, however, show that approximation capability of these networks is not significantly affected despite having strong restriction on weights. The loading of a discrete-weight perceptron is known to be NP-complete [147]. This, combined with the fact that the loading of a 2-layer CWN is NP-complete, makes the task of loading the discrete-weight feedforward network quite formidable. The learning procedure proposed for the training of these discrete-weight networks relies on the modification of the BP cost-function along with a separate discretisation mechanism for 'nearly discrete' weights.

Networks with discrete synapses but continuous offsets have the unique advantage over CWNs in that they do not require a continuous domain multiplier. In particular, if the synapses are restricted to the set $\{-1, 0, 1\}$, the conventional multiplier can be discarded altogether. But how does that affect the approximation capability? It will be analytically shown in Chapter 5 that such networks can approximate functions of one variable with any desired accuracy. Moreover, experiments with multi-variable functions show that this multiplier-free network has approximation capability similar to the CWN. The learning procedure for these networks is a slightly modified version of the one for discrete-weight networks of the last paragraph.

1.12 Historical Note

Artificial multilayer feedforward networks draw their inspiration from the studies on the structure of the biological nervous systems. Such artificial layered structures were first investigated in the late 50's. Their usefulness as iterative learning machines did not manifest itself in that era due to the lack of a suitable learning algorithm [104]. The breakthrough came in 1986 with the publication of two volumes by the PDP group titled *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* [123, 124]. Although ideas similar to the BP learning rule had been presented earlier [79, 109, 154], it became popular only after the publication of these volumes. The next big step in the continued development of the feedforward network was the development of proofs on the universal approximation capability of the 3-layer CWN [80] and later on the 2-layer CWN [29] in the late 80's. The last major achievement in this field, though still in its infancy, has been the integration of CWNs in a Bayesian framework (see, for example [86, 102, 141]). This recent approach about the architecture and learning parameters selection is based on **Bayesian statistics**[23] and holds the promise of delivering optimal generalisation performance.

1.13 Overview of the Thesis

After a presentation of the main theoretical concepts of the work, experiments on a variety of data sets will be used to substantiate the claim that the two constrained-weight networks proposed here are just as effective as conventional networks in practice.

Chapter 2 first provides a simple view of the universal approximation property and then a detailed analytical treatment of a special case of feedforward networks – networks with bounded weights. This bounded-weight result will be required in a later chapter for proving a theorem on the existence of multiplier-free networks. Results focusing on the backpropagation heuristic will then be presented in terms of the relationship between

[23] Bayesian approach differs from the conventional 'frequentist' approach to statistics in that it allows the probability of an event to be expressed as 'degree of belief' in a particular outcome instead of basing it solely on a set of observations [15].

the form of the learning rate and the type of convergence that can be guaranteed for that form. This result will be used in the next chapter in support of the integer-weight learning procedure.

Chapter 3 provides experimental evidence that feedforward networks with low-resolution discrete weights are a viable alternative to networks having continuous weights. These discrete-weight networks, although lacking the universal approximation capability, can implement learning tasks, especially classification tasks, to reasonable accuracies. After highlighting the hardware implementation advantages and justifying the choice of discretisation scheme, this chapter presents one of the main contributions of this thesis – the integer-weight learning procedure. Only the preliminary results on the functionality tests of this procedure will be presented in this chapter. The main results on generalisation performance will be held back until later, where they will be contrasted with those for continuous-weight networks and multiplier-free networks.

Chapter 4 explores the capabilities of discrete-weight networks with the help of the decision and error surfaces of a set of 2-D classification problems, which are generalisations of the conventional 'exclusive or' problem. It is shown that provided a suitable discretisation interval is chosen, a discrete-weight network can be found which performs as well as a continuous-weight network, but that it may require more hidden neurons than its continuous-weight counterpart.

Chapter 5 starts by presenting a new theoretical result on the 'universal approximation in the space of continuous functions of one variable' property of a multiplier-free feedforward network. The multiplier-free functionality of this single-input single-output network has been made possible by restricting the input layer synaptic strengths to $\{-1, 1\}$. This chapter also provides experimental evidence that multiple-input multiple-output multiplier-free feedforward networks with synapses from $\{-1, 0, 1\}$ and unrestricted offsets are a viable alternative to networks having high-resolution weights.

Chapter 6 is concerned with comparing the generalisation performance of the two new feedforward networks proposed in this thesis – integer-weight and multiplier-free – with that of the conventional continuous weight feedforward network. This chapter provides experimental evidence that the new networks, despite having severely constrained

weights, are as effective as conventional feedforward networks. It starts by discussing the techniques that are commonly used for estimating the generalisation performance. Various regularisation techniques used for tailoring the complexity of a network with respect to the learning task at hand are then presented. The target network of the generalisation experiments, the optimal network, is then defined. After presenting the the methodology of the learning experiments, this chapter concludes with the results on three very different data sets.

The final chapter brings together the key ideas presented in this thesis. It also suggests some interesting problems for further work on the 'hardware-friendly' networks proposed here.

Proofs of the theorems and lemmas of Chapter 2 are presented in Appendix A. Appendix B contains the details of the integer-weight learning procedure and the values of the learning parameters used in each of the experiments reported in this thesis. Appendix C displays the decision surfaces mentioned but not shown in Chapter 4. The publications which have resulted from the author's work on this thesis, particularly on the work presented in Chapters 3 and 4, appear in Appendix D. A list of abbreviations and symbols, an extensive glossary, a list of references, and a comprehensive index follow the appendices.

This and the next chapter largely comprise review material, whereas the remaining chapters mainly consist of original work.

2

Theoretical Preliminaries

2.1 Theoretical Questions

For a given artificial feedforward architecture, it is possible to theoretically explore many an interesting issue: What kinds of mappings can it learn (linear, nonlinear, continuous, smooth, measurable) [63, 136]? What type of neuron activation functions are permissible [27, 83, 136]? Is it capable of 'universal approximation' [29, 62, 68]? What is the relation of the quality of approximation with the number of neurons or the depth of weights [22, 64]? How difficult is it to train the network [18]? What is the expected generalisation performance and its relation with the number of necessary training examples [7]? Similarly, questions can be raised about the convergence properties of a given combination of a neural architecture and a specific learning heuristic [74]. This chapter discusses only two of these issues because of their relevance to the work presented in Chapters 3 & 5: a feedforward network's approximation properties and its convergence characteristics with respect to the error backpropagation heuristic.

The theoretical results on the universal approximation property have their importance in the confidence they instil in feedforward network users: they are assured that provided they employ a data set that completely describes the process to be learned,

to train a feedforward network of a suitable complexity with the help of an appropriate learning procedure, they can model the input/output characteristic of *any* mapping that is Borel measurable. Similarly a theoretical analysis can be used to fine tune the convergence characteristics of an individual learning procedure as well as to compare the merits of two competing procedures – all without utilising costly empirical studies.

After providing a simple view of the universal approximation property and a review of the various universal approximation results, this chapter provides a detailed analytical treatment of a special case of feedforward networks – networks with bounded weights. An application of this result will be used to prove the 'multiplier-free network existence' theorem in Chapter 5. It then presents results on the backpropagation heuristic which focus upon the relationship between the form of the learning rate and the type of convergence that can be guaranteed for that form. Some of these results will be used in support of the integer-weight learning procedure presented in Chapter 3.

2.2 Universal Approximation Property of CWNs

The discussion in this section will be restricted to feedforward networks with a single layer of hidden neurons having sigmoidal activation functions, and a single output neuron with a unity activation function. The only property of interest is universal approximation in $C(\mathbb{R}^d)$, the space of continuous functions on $\mathbb{R}^d$.

To start with a simple example, Figure 2.1 shows a continuous function f being approximated with the help of a series of rectangular pulses. The error in the approximation is:

$$\varepsilon = \left[\sum_{m=1}^{M} \left(\int_{x_m}^{x_{m+1}} f(x)\, dx \; - \; f\left(\frac{x_m + x_{m+1}}{2} \right) \Delta x \right)^2 \right]^{\frac{1}{2}}$$

This error, i.e. the cost function, can be decreased to any desired level[1] by selecting an appropriate value for Δx. As $\Delta x = (x_{M+1} - x_1)/M$, the quality of the approximation is proportional to the total number of the pulses in the summation, i.e. M. The approximator generating these pulses is a 'universal approximator' in $C(\mathbb{R})$ if it can

[1] With the exception of identically zero.

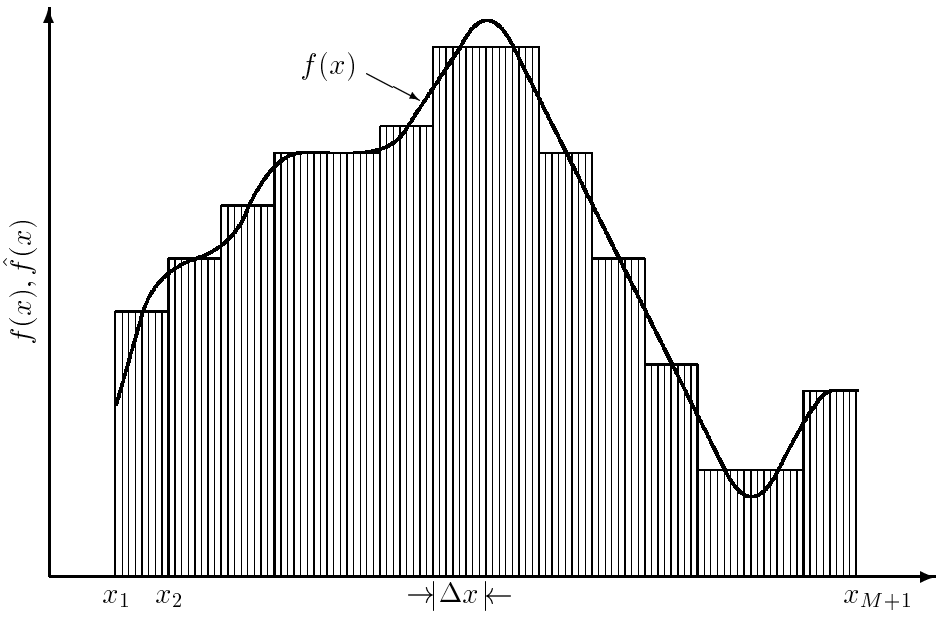


Figure 2.1 Universal approximation in one dimension.

achieve any desired value of ε for any $f \in C(\mathbb{R})$. This requires the approximator to be capable of generating pulses of arbitrary widths, heights, and positions. It can generate such arbitrary pulses if it is capable of generating step functions of arbitrary heights and positions. Therefore, the universal approximating capability of an approximator is solely dependent upon its ability to generate step functions of arbitrary heights and positions. Such arbitrary step functions can obviously be constructed with any desired level of accuracy with functions of the form

$$w^o \sigma(wx + \theta),$$

where $w^o, w, \theta \in \mathbb{R}$, and $\sigma(\cdot)$ is the logistic function. The sum of the magnitudes of the error in this approximation and that in approximating f with rectangular pulses bounds the total error of approximation due to the **triangle inequality**.[2] As both of the component errors can be reduced to any required level, their combination can be reduced to any desired level. This discussion leads to the conclusion that for a suitable value of q, a sum of the form

$$\sum_{j=1}^{q} w_j^o \sigma(w_j x + \theta_j),$$

[2] $|a + b| \leq |a| + |b|$

where $w_j^o, w_j, \theta_j \in \mathbb{R}$, can approximate any function in $C(\mathbb{R})$ to any desired tolerance, i.e. sums of this form are universal approximators in $C(\mathbb{R})$. Similar arguments for approximation in $C(\mathbb{R}^2)$ or $C(\mathbb{R}^d)$ require the capability of constructing 2- (or d)-dimensional 'bricks' of arbitrary sizes at arbitrary positions that can be 'stacked together' to form 2- (or d)-dimensional function.

2.2.1 Universal Approximation in $C(\mathbb{R}^d)$

Numerous results, utilising a vast array of mathematical tools and making various assumptions about the construct of the network, are available on the universal approximation in $C(\mathbb{R}^d)$ property of feedforward networks [26, 27, 29, 41, 55, 62, 63, 83, 136]. Cybenko [29] employed Hahn-Banach and Riesz Representation Theorems, Carroll and Dickinson [26] used Radon transforms for a constructive proof, Funahashi [41] invoked Fourier analysis and Paley-Wiener theory, while Hornik et al. [63] utilised the Stone-Weierstrass theorem to show the universal approximation property with continuous activation functions on **compacta**[3,4] in $C(\mathbb{R}^d)$ with respect to the **supremum**[5] norm.

Leshno et al. [83] showed that non-polynomial activation functions which are locally bounded and piecewise continuous are necessary for universal approximation. Hornik [62] proved universal approximation capability on compacta for networks having activation functions which are locally Riemann integrable and non-polynomial. Chen et al. [27] presented a constructive proof which shows that the sigmoidal activation function need not be monotone or continuous, only bounded for approximation in $\bar{C}(\mathbb{R}^d)$.[6]

All of the above universal approximation results assume that the CWN can have weights of arbitrary precision and magnitude. This assumption is not biologicaly plausible, and is a hindrance from the hardware implementation point of view. Stinchcombe

[3] Closed and bounded subsets of $\mathbb{R}^d$.

[4] Focusing on compact domains in $\mathbb{R}$ instead of the whole of $\mathbb{R}$ makes the problem easier.

[5] Supremum is the least upper bound for a set.

[6] $\bar{C}(\mathbb{R}^d)$ is the extended $C(\mathbb{R}^d)$. If a continuous function is defined in $\mathbb{R}^d$ and $\lim_{|\mathbf{x}|\to\infty} f(\mathbf{x})$ exists, then f is called a continuous function in the extended $\mathbb{R}^d$, i.e. $\bar{\mathbb{R}}^d$, and the set of all continuous functions on $\bar{\mathbb{R}}^d$ is $C(\bar{\mathbb{R}}^d)$ [27].

and White [137] have come up with a result which trades off bounds on the magnitude of network weights with a possible increase in the number of the hidden neurons, which is highly relevant to the work presented in Chapter 5 and will therefore be discussed in detail in the next section.

2.2.2 CWNs with Bounded Weights are Universal Approximators

This section will follow the closely the approach chosen by Stinchcombe and White [137], in which they enforced some restrictions on activation functions to achieve the universal approximation property for networks with bounded weights. The results of this section will be used in proving the multiplier-free network existence theorem of Section 5.2.1.

Only the statements of the main results are presented in this section. The detailed mathematical proofs are provided in Appendix A.

Consider the approximation of functions $f \in C(\mathbb{R}^d)$ with a 2-layer feedforward network $\mathcal{N}^d(\sigma, B)$, where σ is the continuous hidden layer activation function and B, $0 < B < \infty$, is the maximum allowed weight magnitude. Let $\mathbf{A}^d$ be the set of real **affine transforms**[7] on $\mathbb{R}^d$, $A : \mathbb{R}^d \to \mathbb{R}$, i.e. $A(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + \theta$, where $\mathbf{w} \in \mathbb{R}^d$, $\theta \in \mathbb{R}$, and $|A| = \max\{|w_i|, |\theta|\}$. A d-input feedforward network is defined as

$$\mathcal{N}^d(\sigma, B) = \left\{ \hat{f}(\mathbf{x}) = \sum_{j=1}^{q} w_j^o \sigma(A_j(\mathbf{x})) \colon A_j \in \mathbf{A}^d, \text{ and } \max\left\{|w_j^o|, |A_j| \colon 1 \le j \le q\right\} \le B \right\}.$$

where, q is the number of hidden neurons, $\mathbf{x}$ is the d-dimensional input vector, $\mathbf{w}_j$ is the input layer synaptic vector for the j-the hidden neuron, θ_j is the offset and $w_j^o \in \mathbb{R}$ the output layer synapse of that hidden neuron. This network is the same as that of Figure 1.3 except for the restriction on the magnitude of the weights.

A network $\mathcal{N}$ is defined to have the universal approximation property over $C(\mathbb{R}^d)$

[7] Rotations, shifts, and scalings, or any combination thereof, are affine transforms.

if $\mathcal{N}$ is **dense**[8,9] in $C(\mathbb{R}^d)$. $\mathcal{N}^d(\sigma, B)$ is said to be **uniformly dense** in $C(\mathbb{R}^d)$ on the compact set $K \subset \mathbb{R}^d$ if for all $f \in C(\mathbb{R}^d)$ and every $\varepsilon > 0$ there exists $\hat{f} \in \mathcal{N}^d(\sigma, B)$ such that $\sup\{|f(\mathbf{x}) - \hat{f}(\mathbf{x})| : \mathbf{x} \in K\} < \varepsilon$. In this case, $\mathcal{N}^d(\sigma, B)$ is also uniformly dense in $C(K)$. $\mathcal{N}^d(\sigma, B)$ is **uniformly dense on compacta** in $C(\mathbb{R}^d)$ if it is uniformly dense in $C(\mathbb{R}^d)$ on every compact K. The goal of this section is to prove this 'uniform denseness on compacta' property for the CWN with bounded weights.

The following theorem can be used to simplify the investigation into the universal approximation property. It shows that the existence of a universal approximation proof in 1-dimension guarantees its extension in d-dimensions.

Theorem 2.1 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a Borel measurable function and let $0 < B < \infty$. If $\mathcal{N}^1(\sigma, B)$ is uniformly dense in $C(\mathbb{R})$ on some non-empty compact interval $[-s, s], s > 0$, then for $d \in \mathbb{N}$, $\mathcal{N}^d(\sigma, B)$ is uniformly dense on compacta in $C(\mathbb{R}^d)$.* □

The main result of this section is that enforcing bounds on the weights does not destroy the universal approximation capability of a network as long as the activation function is **superanalytic** at some point with a positive radius of convergence. $\sigma \in C(\mathbb{R})$ is defined to be **analytic** at $a \in \mathbb{R}$ with a radius of convergence $r > 0$ if there is an infinite sequence of real numbers, $\{c_n\}$, $n \geq 0$, such that for $|x - a| < r$, $\sum_{n=0}^{\infty} c_n (x-a)^n$ converges and $\sigma(x) = \sum_{n=0}^{\infty} c_n (x-a)^n$. Furthermore, this analytic function σ is defined to be superanalytic at a with a radius of convergence r if for every $n \geq 1$, $c_n \neq 0$. By the next lemma, this superanalyticity property holds if σ is analytic at a with radius r and $c_n \neq 0$ for infinitely many n.

Lemma 2.1 *If σ is analytic at $a \in \mathbb{R}$ with radius $r > 0$, $\sigma(\lambda) = \sum_{n=0}^{\infty} c_n (\lambda - a)^n$, $|\lambda - a| < r$, and $c_n \neq 0$ for infinitely many n then for every b in a dense subset of*

[8] A set A is dense in a set S if $A \subset S$ and $Cl(A) = S$. The **closure** of a set A is defined as $Cl(A) = A \cup \{\text{Limit points of } A\}$. A point p is a **limit point** of A if every neighbourhood of p contains a point $q \neq p$ such that $q \in A$.

[9] For example, rational numbers are dense in irrational numbers. That means between any two irrational numbers there is a rational one or alternatively any irrational number can be approximated within any desired tolerance with a rational number.

$(a-r, a+r)$, *σ is superanalytic at b with radius of convergence* $s = \min\{b-(a-r), (a+r)-b\}$. □

Some examples of functions that are superanalytic almost everywhere are sine, cosine, logistic, and hyperbolic tangent functions. Finite polynomials are examples of functions which are analytic but not superanalytic.

The statement of the main result:

Theorem 2.2 *If for some $a \in \mathbb{R}$, $\sigma \in C(\mathbb{R})$ is superanalytic at a, with radius of convergence $r > 0$, then $\mathcal{N}^d(\sigma, B)$ is uniformly dense on compacta in $C(\mathbb{R}^d)$ for any $B \geq \max\{|a|, 1\}$.* □

The activation functions are required to be superanalytic so that they can be represented as an infinite power series containing *all* powers of x. This 'infinite and complete polynomial' nature of these function makes them dense in all continuous functions.

The next theorem allows further restrictions on the weight values without sacrificing the universal approximation characteristic. The input layer synaptic vectors $\mathbf{w}_j$ can be confined to the unit sphere S^d at the expense of limiting the superanalytic activation functions to those whose derivatives form a basis for the continuous functions. This is due to the Stone-Weierstrass theorem[10] which implies that the set of polynomials in σ – which in the present case is formed by the derivatives of σ – is uniformly dense in $C(\mathbb{R})$.

The following theorem uses the term **span** of a class of functions. For any function $f \in C(\mathbb{R})$ and $r > 0$, $f|(-r, r)$ denotes the restriction of f to the interval $(-r, r)$, and for any set of functions $\mathcal{F}$, $\mathcal{F}|(-r, r)$ denotes $\{f|(-r, r) : f \in \mathcal{F}\}$. For any $\mathcal{F}$ defined on a set $\mathcal{O}$, the span of $\mathcal{F}$, $sp(\mathcal{F})$, denotes the closure of the set of finite linear combinations of elements of $\mathcal{F}$ in the topology of uniform convergence on compact subsets of $\mathcal{O}$.

Theorem 2.3 *If the conditions of Theorem 2.2 hold and $\mathbf{w}_j \in S^1 \cup \{0\}$ then $\mathcal{N}^1(\sigma, B)$ is uniformly dense on compact subsets of $(-r, r)$ in $sp(\{\sigma^{(k)}|(-r, r) : k \geq 0\})$ for*

[10] See Appendix A for a statement of this theorem.

any $B \geq 1$ *where* $\sigma^{(k)}$ *is the* k*-th derivative of* σ*. If in addition* $sp(\{\sigma^{(k)}|(-r,r) : k \geq 0\}) = C(\mathbb{R})|(-r,r)$ *then for* $\mathbf{w}_j \in S^d \cup \{0\}$, $\mathcal{N}^d(\sigma, B)$ *is uniformly dense on compacta in* $C(\mathbb{R}^d)$. □

Theorem 2.3 states that the superanalyticity requirement for σ, although necessary, is not sufficient for universal approximation if synapstic vectors in the input layer are restricted to the unit-sphere. Consider the case of sine function, which is superanalytic everywhere except at 0, but the span of its derivatives is not dense, and therefore uniform approximation by $\mathcal{N}^d(\sigma, B)$ with input-layer synapses on the unit-sphere fails. This can be easily illustrated in the 1-dimension case. Let $\sigma(\cdot) = \sin(\cdot)$ and consider the set

$$\mathcal{N}^1(\sin, S^1) = \left\{ \hat{f} : \hat{f}(x) = \sum_{j=1}^{q} w_j^o \sin(\pm x + \theta_j),\ 1 \leq j \leq n,\ n \in \mathbb{N},\ \theta_j \in \mathbb{R} \right\}.$$

By the addition formulas for the sine function, any such $\hat{f}$ must be of the form $c_1 \sin(x) + c_2 \cos(x)$ for some $c_1, c_2 \in \mathbb{R}$. This set of functions, $\mathcal{N}^1(\sin, S^1)$, is clearly not dense on compacta in $C(\mathbb{R})$, being unable, for example, to universally approximate $\sin(20x)$ on $[-1, 1]$.

An application of Theorem 2.3 provides the following very useful result:

Lemma 2.2 *If* σ *is the logistic function then for* $\mathbf{w}_j \in S^d \cup \{0\}$, $\mathcal{N}^d(\sigma, B)$*is uniformly dense in compacta in* $C(\mathbb{R}^d)$.

A similar application of Theorem 2.3 will be used in Chapter 5 to prove the universal approximation property of the multiplier-free net.

2.3 Convergence of Error Backpropagation

The error backpropagation heuristic for training feedforward networks, the method of choice for many users, is not a completely new training procedure: the batch version is known as (total) gradient descent [94], the on-line version is familiar to statisticians as stochastic gradient descent [156], and the momentum modification of the batch version can be recognised as the 'heavy ball' method discussed in the numerical analysis

literature [126]. The convergence characteristics of BP can be empirically analysed by the Monte Carlo techniques [122] or theoretically by various statistical and numerical analysis tools. The theoretical results presented in this section will comment on the form of permissible cost functions, and the effect of the learning parameters on convergence to a solution.

Tesauro et al. [139] have examined the asymptotic behaviour of the batch version of BP in terms of the decrease in error with respect to the number of epochs t. This study found the error to decrease as $1/t$ for large t. It is also shown that best convergence was achieved using logistic activation function in the hidden layer. In terms of learning parameters, it was found that for a constant learning rate, the rate of convergence is unaffected by the addition of a momentum term or addition of the **margin heuristic**[11] to the BP learning procedure. Moreover, the results showed that convergence rate can be increased by adapting the learning rate with respect to the second derivative of the error surface.

From the classical steepest descent results, it is known that the batch version of BP converges to a solution for a fixed learning rate. The on-line version with a fixed learning rate, however, starts to wander around the error surface as the error becomes small and does not settle into a stable solution. A diminishing learning rate is the weakest requirement for on-line BP to settle down [155]. White [155] views on-line BP as 'an application of Robbins-Monro stochastic optimisation procedure [118] to solve the first order conditions for a nonlinear least squares regression problem'. He considers a sequence of independent, identically distributed, and bounded training examples, and bounded and twice continuously differentiable hidden layer activation functions. This study concludes that on-line BP **converges with probability 1**[12] for successive values of the learning rate $\eta_t, \eta_{t+1}, \ldots$ if $\eta_t \propto t^{-\nu}$, $0 < \nu \leq 1$.

Kuan and Hornik [74] investigated on-line BP learning with a small, constant learn-

[11] Error in the output of a neuron is not backpropagated if it is within a specified small margin.

[12] If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges with probability 1 to a, i.e. $\hat{a}_n \to a$ as $n \to \infty$ or $\hat{a}_n \xrightarrow{P=1} a$, if $\lim_{a_n \to \infty} P[\hat{a}_n = a] = 1$ for some real number a. Also known as almost sure convergence, convergence almost everywhere, and strong convergence [155].

ing rate. They considered the weight update steps as an interpolation process and analysed this process by associating an ordinary differential equation (ODE) with it and by studying the asymptotic behaviour of the solutions of the ODE. The asymptotic study of the behaviour serves only as a 'first order approximation' to the algorithm. They made two assumptions about the learning procedure: first, all training examples are bounded and the sequence of examples is **strongly stationary**[13] and **ergodic**.[14] Secondly, the cost function is continuously differentiable with respect to weights and training examples.[15] This result of this study indicates **convergence in probability**[16] for the BP procedure. This convergence in probability result is weaker than the 'convergence with probability 1' result, and means that convergence to optimal weights in most training runs is expected but not guaranteed in general for on-line BP with a small constant learning rate [74].

The discussion in this section has so far been about networks converging to *a* minimum in the error surface: this minimum can be a local one or a global minimum. A modified version of on-line BP with a diminishing learning rate has been proposed for finding global minima, the modification being the addition of decreasing amounts of random noise to the weights at each weight modification step [75]. This procedure is similar to simulated annealing and is guaranteed to converge with probability 1 to a global minimum.

2.4 Summary

The two key results of this chapter were:

[13] If a random variable X is strongly stationary then the distribution of $X(t)$ is the same for all times t [45].

[14] A random process is ergodic if its ensemble and temporal averages are the same.

[15] The cost function, e.g. the one shown in Equation 1.1, depends upon the output of the network, which, in turn, is a function of the activation function $\sigma(\cdot)$, which transform the $\mathbf{x} \cdot \mathbf{w} + \theta$ terms. This condition is satisfied if all $\sigma(\mathbf{x} \cdot \mathbf{w} + \theta)$ terms are continuously differentiable for all $\mathbf{x}$, $\mathbf{w}$, and θ.

[16] If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges to a in probability, i.e. $\hat{a}_n \xrightarrow{P} a$, if there exists a real number a such that for any $\epsilon > 0$, $P[|\hat{a}_n - a| < \epsilon] \rightarrow 1$ as $n \rightarrow \infty$. Also known as weak convergence [155].

First, a feedforward network having hidden layer activation functions that are superanalytic and these superanalytic activation functions having derivatives that form a basis for the continuous functions, can have input layer weight vectors restricted to the unit-sphere without sacrificing its ability to approximate all continuous functions to any desired degree of accuracy.

Secondly, the BP learning heuristic with a small constant learning rate converges in probability to a solution as long as the cost function is continuously differentiable with respect to weights and training examples.

3

Integer Weight Networks

3.1 Why Integer Weights?

Compared with their continuous-weight counterparts, feedforward networks having integer synapses and offsets are easier to implement in electronics as well as in optics. For example, integer weights in the range $[-3, 3]$ can be represented by just 3 bits. This property reduces the amount of memory required for weight storage in digital electronic implementations. It also simplifies the digital multiplication operation as multiplying any number with this 3-bit weight requires a maximum of three basic instructions – one shift, one add, and one sign-change. Moreover, if the inputs are restricted to the set $\{-1, 1\}$, the neurons in the first hidden layer require only sign adjustment for multiplication operations, and only integer addition. The activation function of these neurons, e.g. hyperbolic tangent, can be accurately[1] implemented by using a lookup table with only 4 entries augmented by sign adjustments. For binary bipolar inputs, the addition and multiplication operations in the output layer neurons of a 2-layer network are not as simple as that for the hidden ones, but can be streamlined, keeping in mind the limited number of possible output levels of the hidden neurons.

[1] With an approximation error which is always less than 0.5%.

In analogue implementations, where weights can be stored as resistances, the integer weight scheme requires only 3 distinct resistance values, which streamlines the manufacture by avoiding the expensive trimming schemes required to achieve the precise resistances for conventional networks. The presence of zero valued weights also simplifies the design of analogue hardware.

In optically implemented networks, where weight values can be represented as grey-scale masks or voltage levels for spatial light modulator, the complexity of implementation is again reduced because of integer valued weights.

These features of the Integer-Weight Network (IWN) make it very attractive for efficient hardware implementation. It should, however, be stressed here that these benefits are valid only when the IWN has been trained, as the learning task still requires high-resolution arithmetic. This makes the IWN unsuitable for in-situ learning. Moreover, a high-resolution activation function is required during training and for the trained network.

IWNs may have some performance related benefits besides the obvious hardware implementation advantage. For example, the integer-weight learning procedure proposed in this chapter generates networks with some weights having a value of zero. The number of these zero-valued weights is higher if larger-than-optimal IWNs are used for training. The reduced number of effective weights, combined with the fact that all non-zero weights are restricted to only six values, limits the complexity of the network. This can result in reduced over-fitting and therefore improved generalisation performance: it is known from **shrinkage**[2] estimation and **ridge regression**[3] analysis in linear models that generalisation can be improved by reducing the size of the weights from estimates that give best fit in the sample [128]. The case of restricting the weights to small integers is most similar to **penalised ridging**, in which the cost function is

[2]The difference between training set accuracy and the test set accuracy.

[3]The precision of least-squares estimates gets worse with an increase in dependence between the input variables. Ridge regression estimators are more precise in those situations and are obtained as the estimators whose distance to an ellipsoid (the 'ridge'), centred at a least-squares estimate from the origin of a parameter space, is a minimum [46].

Table 3.1 Weight resolution terminology

Weights	Bits
Low resolution	1–4
Medium resolution	5–16
High resolution	16+

augmented by a penalty term[4] [130].

Another potential advantage of the IWN is its relatively immunity to noise in training data. The IWN only captures the main features of the training data in its discrete-valued weights. The low amplitude noise present in the training data cannot perturb the discrete weights, because these weights require relatively larger variations in order to jump from one discrete value to another[5]

This chapter provides experimental evidence that feedforward networks with low-resolution discrete weights are a viable alternative to networks having continuous weights. These discrete-weight networks, although lacking the universal approximation capability, can implement learning tasks, especially classification tasks, to reasonable accuracies. After highlighting the hardware implementation advantages and justifying the choice of discretisation scheme, this chapter presents one of the two main contributions of this thesis – the integer-weight learning procedure. Only the preliminary results on the functionality tests of this procedure will be presented in this chapter. The main results on generalisation performance will be held back until Chapter 6, where they will be contrasted with those for continuous-weight networks and multiplier-free networks.

3.2 Discrete-weight Networks

Most of the experimental work on feedforward networks is based on weights of high resolution (see Table 3.1). There has been some work on networks with medium resolution

[4]The other two types of ridging are **constrained ridging**, in which some norm of the weights is constrained to a specific value, and **smoothed ridging**, in which noise is introduced in the inputs [130].

[5]Examples indicating such a behaviour will be presented in Section 6.5.2.

weights but very little on ones with low resolution weights. Various claims have been made about the minimum b_w required to learn problems of varying complexity, but no generally agreed standards exist [33, 58, 60, 77, 88, 89, 112, 138, 148, 166, 167, 169]. Most workers, however, agree that networks with weights of medium resolution are adequate to model many learning tasks to the desired accuracy [33, 169], [58, 60, 112, 166, 167][6] . The networks with low resolution weights have mostly been considered by the workers who were interested in streamlining the multiplication operation in neurons. They considered weight values that were either powers-of-two [77, 138][6] or sum-of-powers-of-two [88, 89][6]. For such weights, multiplication operations can be replaced by much simpler shift operations or a sequence of sum-and-shift steps, respectively.

Results on some very simple leaning tasks with feedforward networks having binary and ternary weights were presented by Perez Vincente et al. in [148]. There is, however, a wealth of literature available on single-layer perceptrons with binary weights (e.g. [147]).

3.3 The Hidden Neurons -vs- Weight-Depth Trade-Off

In selecting the architecture (i.e. complexity) of a feedforward network, the designer has to make choices about two information resources – the number of hidden neurons, q, and weight depth, b_w. The combination of these two must match or exceed the complexity of the learning task at hand. In general, the two can be traded off with one another to achieve a desired level of error at the output of the network. This trade off process, however, does have its limits – one neuron with weights of infinite resolution cannot implement most mappings, and a network with only single-bit weights may require a prohibitively large number of neurons to implement many mappings. Therefore, the combination of q and b_w, the network complexity, is not a linear function of its components [22]. The optimal combination of these two parameters to implement a desired network complexity may or may not be unique. General methods to find the optimal combination are not currently available. An attractive option would be to

[6]The results of these papers were based on discrete weights as well as discrete activation functions.

dynamically vary q and b_w during training to find their optimal values. Currently, many sets of heuristics exist that dynamically modify q (e.g. Cascade-correlation [37]), but the author is unaware of any work in which just b_w, or both q and b_w simultaneously, were varied during training.

3.4 Approximation Capabilities

It has been pointed out by Wray and Green [165] that the very fact that feedforward networks are implemented in software/hardware on computers of finite word-lengths implies that the hidden-neuron activation functions are finite polynomials,[7] and are therefore not capable of universal approximation, as was shown by Leshno et al. in [83]. Does this statement mean that the universal approximation results of Section 2.2 do not have any practical significance and are useful only as a theoretical curiosity? The significance of the universal approximation results is in the guarantee they provide – for any desired degree of precision, a realisable network exists that can achieve that degree of precision. The choice of machine word-length limits only the upper limit on the achievable degree of precision.

The choice of machine word-length controls both b_w and the precision of the network's activation functions. It can be demonstrated that even if a network's activation functions *were* non-polynomials, the limitations imposed by bounded discrete weights are enough to destroy its universal approximation capability. This can be easily demonstrated by the argument that the set of basis functions,

$$\mathcal{Z} = \{w_i^o \sigma(\mathbf{w}_i \cdot \mathbf{x} + \theta_i)\}$$

where $w_i^o, w_i, \theta_i \in \{\text{finite subset of } \mathbb{Z}\}$, is a *finite* set. As $\mathcal{Z}$ is finite, it cannot be dense in $C(\mathbb{R}^d)$ for any d. Hence:

An IWN, or any feedforward network having bounded and discrete weights only, cannot *be a universal approximator.*

[7] In the case of analogue computers, their inherent inaccuracy due to imperfections in fabrication and fluctuation in operating conditions result in similar problems [106].

This means that IWNs may not be able to learn every given mapping with arbitrary accuracy. This is quite a drawback! On the other hand, it is always useful to try to model a given mapping with integer weights. If the results do meet the desired level of accuracy then all the benefits of networks having low resolution weights are at hand. If, however, the results are not as desired then weights of a finer resolution can be employed. This increase in resolution can be repeated to obtain the desired level of performance. The increase in resolution and non-integer discrete weights will be further discussed in Chapter 4.

The above discussion points towards the suitability of IWNs for learning pattern-classification tasks as opposed to function approximation ones. This is due to the fact that the error tolerance requirements for function approximators are usually less flexible as compared with those for pattern classifiers.

3.5 Learning Heuristics

It was stated in Section 1.6 that loading a perceptron with integer weights is NP-complete and loading a fixed architecture CWN is also NP-complete, hence loading an IWN should be very difficult, which indicates that the mean number of learning epochs for IWNs should be quite large compared with that expected for the comparable CWNs.

While selecting a set of heuristics for training a network, a choice has to be made between gradient descent and stochastic techniques, as was discussed in Section 1.8. For CWNs the choice is usually quite clear because gradient descent has been found to be quite effective on most learning tasks discussed in the literature. Stochastic techniques are generally used for learning tasks for which the error surfaces are perceived to be particularly 'choppy' and full of local minima. The other advantage of stochastic methods is that they do not involve the computation of gradients, which from the hardware implementation point of view, can be a very expensive task.

The integer-weight learning heuristics proposed in this chapter are mainly based on gradient descent. In the initial versions of the collection, they were just that, but later some stochastic flavour was added to enhance the probability of convergence to an

acceptable solution. Other modifications were also applied to improve implementation efficiency. The feature unique to the design of discrete-weight networks is the selection of the weight discretisation scheme: the discretisation can be either uniform or non-uniform. It can be selected before training, during training, or at the end of training. A choice has also to be made about the initialisation of weights: should they be discrete or continuous? In either case, what should be their initial distribution?

These choices, combined with the choices about the size of the hidden layer, the neuron activation function, and the frequency of weight updates, make the task of designing efficient learning heuristics particularly difficult. Keeping this in mind, it was decided from the very beginning to concentrate on designing heuristics that will get the job done in a reasonable number of epochs and the task of achieving the fastest convergence times was therefore not considered at all.

3.5.1 Weight Discretisation Schemes

Networks having powers-of-two weights represent an interesting discretisation scheme for digital implementations [76, 77, 138]. They have the advantage in arithmetical manipulations – multiplying a number with a power-of-two weight can be accomplished by a simple shift operation on that number, which is faster and requires much less hardware than the conventional digital multiplier. In this scheme, only the synapses have to be power-of-two, the offsets can still be real numbers as they are never involved in multiplication operations. A variation on this scheme – simple sums of powers-of-two weights – can provide better resolutions while somewhat maintaining the ease of multiplication: multiplying a number with a sum-of-power-of-two weight can be accomplished by a short sequence of simple shift-and-add operations [88, 89]. These two schemes and the related learning procedures were inspired by similar hardware streamlining work on FIR filter design [170].

Another possible avenue is to take a trained CWN, and analyse its weight distribution to find any clusters. After determining an appropriate number of clusters depending upon the desired number of discretisation levels, the weights in each cluster are assigned to the centre value of the cluster. The discretised network is

now tested and if the performance is below par, then it is trained again and the clustering→discretisation→testing→training sequence is repeated until a satisfactory result is achieved. This method results in weights which are non-uniformly discretised through a problem dependent multiple-thresholding discretisation function. This technique, *sans* clustering[8] was evaluated by Fiesler et al. in [40]. The authors claimed their results to be quite encouraging. They did not, however, provide much information about the complexity of the benchmark learning tasks.

The most popular learning scheme is just to use medium resolution weights throughout training [33,58,60,112,166]. In this scheme the network is trained with a preselected b_w on the target learning task. If learning to a desired level of accuracy is not accomplished then training is restarted with an increased b_w.

The prototype set of low resolution weights used in this chapter is $\{-3, -2, -1, 0, 1, 2, 3\}$. The choice of integers as discrete weights was made due to their ease in arithmetical manipulations, and the range was restricted to 3 because it required the lowest number of bits for which trained networks gave acceptable performance on a set of benchmark learning tasks. In the preliminary experiments, networks with 1-bit and 2-bit weights did not learn the target tasks to acceptable tolerances, and therefore were eliminated from further study. The weight set $\{-2, -1, 0, 1, 2\}$ was not considered because it required 3 bits/weight for storage without utilising the full dynamic range of 3 bits. The eighth available level in 3-bits was not used because of the fear that an asymmetric set of weights may result in poor learning performance. It should be noted here that the selected set of integer weights are easily expressible by simple sums of powers-of-two, and therefore require at most a single shift, a single sum, and a single sign-change instruction to accomplish the multiplication operation.

3.5.2 Discrete-weight Learning Techniques

The simplest approach taken for forming networks with discrete weights is to truncate bits from the weights of a network trained with high resolution weights. This is based on the suggestion that numerical algorithms require temporary 'guard bits' on each weight

[8] They used a static and uniform discretisation (stair-case like) scheme instead.

during training, and these guard bits, though essential during training, do not affect the network's performance if removed after it has been fully trained [30]. Based on this idea, a multiple-thresholding method has been proposed for generating discrete-weight networks [28, 164]. In this simple method, the continuous weights of a fully trained network are discretised using a staircase function. This technique was used by Chieueh and Goodman for training with ternary weights [28] and it was found that a large percentage of the resulting networks failed to perform correctly when the weights were discretised. This was a rather drastic example of the truncation technique. Truncating the last bit or two of high-resolution weights should give more reasonable results.

The Continuous-Discrete Learning Method [40] follows a more fruitful strategy. In this method, a trained CWN is discretised according to a predetermined stair-case function, and then trained again using the standard BP procedure. The discretisation-training cycle is repeated until the network converges to a satisfactory solution. The authors claimed the results to be quite encouraging. They did not, however, provide much information about the complexity of the benchmark learning task. The fundamental assumption behind this learning scheme is that the locations of acceptable minima for discrete-weight networks and CWNs are the same. This author is, however, unaware of any evidence, theoretical or experimental, which supports this assumption.

The method put forward by Marchesi et al. [88, 89] for generating networks with powers-of-two or sum of powers-of-two weights and unrestricted offsets uses a trained CWN as the starting point. Its weights are discretised and then a modification of BP is used for further training: the modification being that the weights are updated only if the new weight is also an allowed discrete value. This training method will surely get trapped in shallow local minima because the update rule does not account for a long sequence of very small BP weight changes.

Weight perturbation is an alternative to the BP learning procedure and can be used for discrete-weight learning. In this method, all of the weights are perturbed in turn and the associated change in the output of the network is used to approximate local

gradients [59]. The weight update rule for this technique is:

$$\Delta w = -\eta \frac{E_{o_{initial}} - E_{o_{perturbed}}}{w_{initial} - w_{perturbed}} = -\eta \frac{\Delta E_o}{\delta w} \tag{3.1}$$

This technique, although not as efficient as BP, requires only feedforward calculations for its operation, which simplifies its implementation in hardware [166]. It does, however, have two major disadvantages: a) the effect of changes in the input layer weights on the output is relatively small and therefore difficult to measure accurately; b) the computational complexity depends more strongly on size than in standard BP [33] as E_o has to be recalculated after each weight is updated. A modification of this method, the Chain Rule Perturbation (CHRP) technique [59], overcomes both of these drawbacks. This technique uses Equation 3.1 for updating output layer weights. It then perturbs a hidden neuron output by δu to determine $\Delta E/\delta u$. All weights feeding into that hidden neuron are then simultaneously perturbed to compute $\Delta u/\delta w$. The combination of these two measurements is then used to update the hidden-layer weights as follows:

$$\Delta w_{ij} = -\eta \frac{\Delta E_o}{\Delta w_{ij}} = -\eta \frac{\Delta E_o}{\delta u_j} \frac{\Delta u_j}{\delta w_{ij}} \tag{3.2}$$

CHRP is advantageous in that the two derivatives of Equation 3.2 can be calculated more accurately compared with the only derivative of Equation 3.1, especially when that derivative has a small value. It also has the added benefit of semi-parallel operation compared to the strictly serial updating procedure of Equation 3.1 [59]. For learning with discrete-weights, this method can be restricted to take discrete steps only. This learning technique is very suitable for on-chip implementations. It lacks the mathematical efficiency of BP however, and therefore requires a large number of epochs to reach acceptable solutions.

The 'backpropagation with quantisation' technique, put forward by Dundar and Rose, uses BP for learning until the network reaches the 'paralysed' stage of a local minimum – the BP weight updates are too small to change weights from one discrete value to another. It then 'jolts' the network out of this condition by incrementing weights with the larger updates to the next discrete level [33] depending upon the comparison between the sum of the last k updates and a user selectable threshold. This

scheme needs to be augmented with some stochastic mechanism because in its current form, although it does take care of the static paralysis i.e. no changes in weights, it could result in a dynamic paralysis in which weights are 'jolted' back and forth between two values.

Stochastic techniques like simulated annealing [69] can also be used to train CWNs or discrete-weight networks. These techniques have the advantage of global optimisation – they search for global minima in error surfaces, and do not get stuck in local minima, as BP can. Stochastic techniques have the added advantage that they do not require gradient computations, which is very attractive from the hardware implementation point of view. The main drawback of this class of techniques is their speed. They generally require a large number of epochs for convergence and each epoch generally requires the recalculation of E_o for every training example and every weight modification.

Combining the gradient-descent based and stochastic steps can also be fruitful. The Combined Search Algorithm [166] algorithm is an example of such an approach. It uses BP to search for solutions, and stochastic steps to jump out of shallow local minima. The stochastic step, termed as the Partial Random Search (PRS), involves the replacement of a single weight with a random number selected uniformly from a predetermined range. If this replacement decreases the output error then it is made permanent; otherwise the weight is restored to its original value. The main problem with this method is that every single weight change caused by PRS requires the time consuming recalculation of E_o for the whole network.

Hoehfeld and Fahlman used the gradient-descent based Cascade-correlation ontogenic method combined with probabilistic rounding for training networks with medium-resolution weights [58]. If the Δw calculated by gradient-descent was smaller than the minimum discrete step then the relevant weight was incremented according to a probability proportional to the size of Δw. This technique showed promising results on test problems of reasonable complexities. Those results were, however, for medium-resolution weights only.

All of the above methods either can start or do start with a trained CWN as the

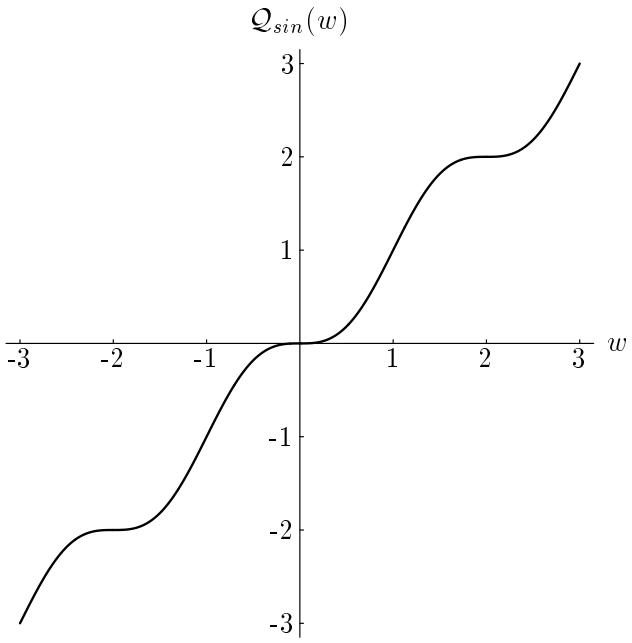
initial point. In starting with a trained CWN, the designer is hoping that the error minimum found for the CWN is the same, or is very close to, the one for the desired discrete weight network. This may or may not be the case in general, especially for networks with complex error surfaces. In such cases, it is better to start looking for a discrete solution from the very beginning of the training phase. A start can be made with an untrained network with its weights randomly initialised to small discrete values according to some random distribution and use one of the many stochastic optimisation techniques for training. This procedure is very useful for those silicon implementations which require on-chip learning, because the absence of continuous weights and of backward propagation of errors simplifies the design immensely.

The integer-weight learning procedure presented in this chapter uses the gradient descent heuristic for both output error minimisation and weight discretisation. The procedure minimises both error in the output of the network, E_o, and the weight discretisation error, $\boldsymbol{E_w}$, simultaneously. Each weight modification is a simple sum of the independently calculated updates which minimise each of the two errors. The learning cycle is stopped only when E_w becomes zero and E_o reaches an acceptable level.

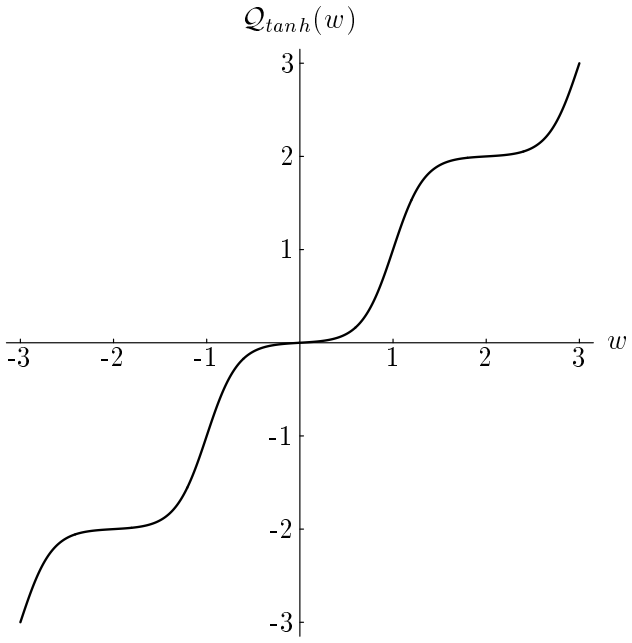
3.5.3 Integer-weight Learning

An outline of the new integer-weight learning procedure will now be presented. The procedure starts by initialising the network with small continuous weights selected according to a uniform random distribution. The conventional on-line BP algorithm is used for the minimisation of the error in the network's output. A new weight discretisation mechanism is superimposed on this BP procedure. This weight discretisation mechanism minimises the difference between weights and a static weight discretisation function $\mathcal{Q}(w)$ in the mean-squared sense. Therefore, the combined error function to be minimised is:

$$\begin{aligned} E(\mathbf{W}) &= E_o(\mathbf{W}) + E_w(\mathbf{W}) \\ &= \sum_{\substack{\text{all outputs} \\ \text{all examples}}} [t-o]^2 + \sum_{\text{all weights}} [\mathcal{Q}(w)-w]^2 \end{aligned} \tag{3.3}$$



$$\mathcal{Q}_{sin}(w) = w + \frac{\sin(w+1)\pi}{\pi}$$



$$\mathcal{Q}_{tanh}(w) = \sum_{i=0}^{3} \tanh[3\,(x+2i-3)]$$

Figure 3.1 Two choices for the discretising function $\mathcal{Q}(w)$ [84].

Table 3.2 Integer-weight learning

Initialise weights in the range $(-0.5, 0.5)$

Select example

  Select a weight, w

    $w \leftarrow w - \eta \frac{\partial E_o}{\partial w}$

    $w \leftarrow w - \chi(\mathcal{Q}_{tanh}(w) - w)\left(\frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w} - 1\right)$

  Loop

Loop until $E_o < \varepsilon$ and $E_w = 0$

The choice of $\mathcal{Q}(w)$ is critical for the efficient discretisation of weights. The two choices that were considered are shown in Figure 3.1. The key feature of these functions is that the zeros of $(\mathcal{Q}(w) - w)$ have the required integer values. In practice, the application of these functions is restricted to the interval $[-3, 3]$, since any weight values outside this interval are truncated to $\{-3, 3\}$. Of the two, $\mathcal{Q}_{tanh}(w)$ is computationally more expensive to generate, yet it does have the advantage in having an adjustable slope between discrete values. It also uses the same hyperbolic tangent function as the neuron activation function.

The weight modification that can be used to minimise the combined error function is:

$$\Delta w = -\Delta w_o - \Delta w_w \tag{3.4}$$

$$\Delta w_o = \eta \frac{\partial E_o}{\partial w} \tag{3.5}$$

$$\begin{aligned} \Delta w_w &= \chi \frac{\partial E_w}{\partial w} \\ &= \chi\left(\mathcal{Q}_{tanh}(w) - w\right)\left(\frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w} - 1\right) \end{aligned} \tag{3.6}$$

where χ is the **weight discretisation rate**. A summary of the integer-weight learning procedure is shown in Table 3.2.

3.5.4 Convergence Properties

The above described integer-weight learning procedure is a modified version of the conventional on-line BP procedure, the modification being the addition of a nonlinear weight shaping function to the objective function. This modified objective function contains a combination of hyperbolic tangent functions only. As the hyperbolic tangent function is continuously differentiable, all the discussion of Section 2.3 applies. Hence, integer-weight learning with a constant learning rate is guaranteed to converge in probability to a solution provided that the sequence of training examples is strongly stationary and ergodic. Convergence with probability 1 can be achieved by using a sequence of diminishing learning rates.

3.5.5 Practical Considerations

The results on the initial trials with the integer-weight learning procedure were not very promising. Many of the trial runs failed to converge even on simple learning tasks. The number of epochs required for convergence to a solution was unacceptably large because the weight values hovered around discrete levels for too long without actually reaching them. This problem was mainly caused by the interaction of the error minimisation and the discretisation mechanisms – the two mechanisms were nullifying each other's weight modifications which was resulting in learning paralysis. The learning procedure was modified to improve its convergence efficiency. The following set of guidelines was drafted for the design of the learning procedure keeping in mind the results of the initial experiments:

> The procedure must mainly be based on gradient descent to benefit from the faster speed of that heuristic. It should, however, have a stochastic component to account for the large number of local minima observed during the initial experiments. The paralysis observed during the initial stages of learning should be avoided by letting the BP mechanism dominate the learning process when E_o is large. The weight-discretisation mechanism can have the upper hand when E_o is small. The weight-discretisation process

should be augmented with a weight-rounding mechanism to overcome the slowness of convergence when weights have 'nearly discrete' values. Lastly, the non-critical but time consuming exact calculations should be replaced by their approximate but faster incarnations.

Keeping these goals in mind, it was decided that the learning process should start with little attention to weight discretisation: the discretisation mechanism should slowly come into play as the network starts moving towards a solution, and should become stronger with progress in learning. This was accomplished by computing a new E_o-dependent χ before each learning epoch: χ was made exponentially dependent upon the negative of E_o.

$$\chi := \alpha_\chi \; e^{(\varepsilon - E_o)\beta_\chi} \tag{3.7}$$

where α_χ and β_χ are empirical parameters, and ε is the maximum acceptable value of E_o. This strategy did help in the initial stage of training but not during the intermediate stage. The most likely reason was the formation of new local minima due to discretisation. It is well known that the standard BP algorithm sometimes gets stuck in local minima [122]. The superposition of the weight discretisation process on BP can result in changes in the shapes of these minima [166] or even an increase in their number. At the start of discrete-weight learning, E_o is large, and the output error minimisation process dominates. Conversely, the weight discretisation process has the upper hand when E_o is small. At intermediate values of E_o, the two processes may nullify each other's effect – resulting in a local minimum. To avoid this type of local minimum, the weight discretisation process was augmented by a random perturbation mechanism:

$$\Delta w_w \leftarrow \Delta w_w \tan(RND), \tag{3.8}$$

where RND is a random number selected uniformly from $(0, \pi/2)$. $\tan(RND)$ has a small value most of the time but infrequently it can become very large (see Figure 3.2). This perturbation strategy is akin to shaking a marble enclosed in a sealed box having an unknown landscape, with the goal to move the marble to the global minimum in

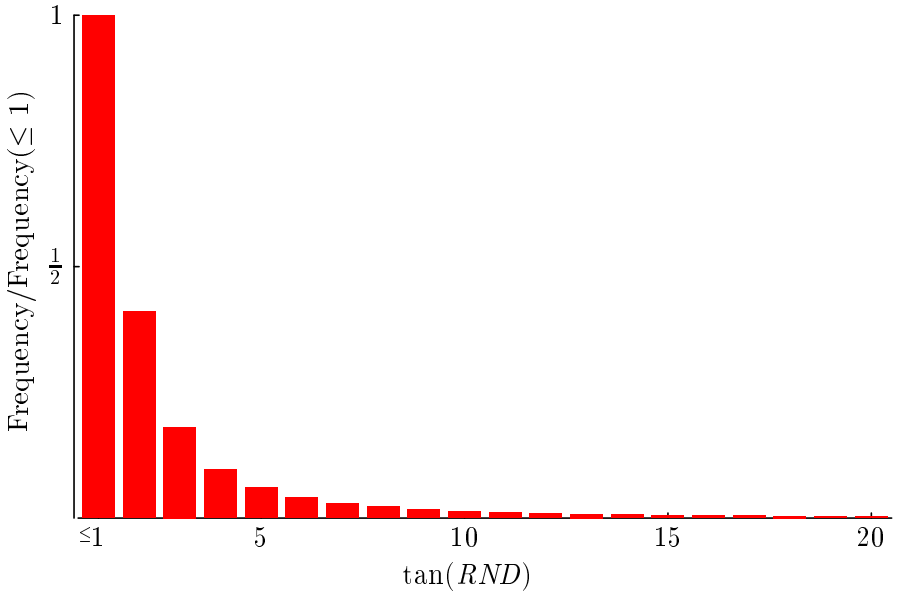


Figure 3.2 Frequency distribution of numbers generated by tan(*RND*).

the landscape without peeking inside the box[9] One way of accomplishing this goal is to shake the box gently most of the time and vigorously for short bursts only. Shaking gently will move the marble out of shallow local minima, and will not have any effect if the marble is residing in one of the relatively deeper minima. A vigorous shake every once in a while, will dislodge the marble from all but the deepest minimum. At the end of many such gentle-vigorous shaking cycles, the marble will, with high probability, come to rest in the global minimum.

The tan(*RND*) perturbation strategy was successful, except in cases where a discrete-weight network with an unacceptably large E_o was generated. The solution that was adopted in those cases was to strengthen the E_o minimisation process. This was accomplished by temporarily boosting the value of η for one learning epoch.

The weight-discretisation process uses a mean-squared error minimisation heuristic. The progress of this method is dependent upon the magnitude of the error. This progress becomes painfully slow as the system becomes closer to a solution. To overcome this problem, a rounding mechanism for weights with 'nearly integer' values was added

[9]This 'marble in a box' analogy is adapted from [43].

to the discretisation process. This mechanism acts as a set of 'black holes' centred at each integer value. If a weight falls within the **black hole radius**, ρ, its value is forced to the centre value of the black hole. The radius was computed before each learning epoch, and was made exponentially dependent upon the negative of E_o.

$$\rho := \alpha_\rho \, e^{(\varepsilon - E_o)\beta_\rho} \tag{3.9}$$

where α_ρ and β_ρ are empirical constants.

As a finishing touch, the $(\mathcal{Q}_{tanh}(w) - w)\left(\frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w} - 1\right)$ term was replaced by an easily computed approximation $(w - \mathcal{Q}_{prac}(w))$ (see Figures 3.3 and 3.4). Easier computation is not the only advantage of the approximate term – it also provides stronger 'pull' when a weight value is midway between two discrete level. On the other hand, this approximation is contrary to the requirement of the convergence guarantee of Section 3.5.4.

After making all of the above changes the final version of the Δw_w modification is:

$$\Delta w_w = \alpha_\chi \, e^{(\varepsilon - E_o)\beta_\chi} (w - \mathcal{Q}_{prac}(w)) \tan(RND). \tag{3.10}$$

A momentum term was added to the BP mechanism to strengthen E_o minimisation throughout the learning process.

$$\Delta w_o = \eta \frac{\partial E_o}{\partial w} \; - \; \mu \Delta w_{o_{old}} \tag{3.11}$$

where $\Delta w_{o_{old}}$ is the Δw_o calculated in the previous step. These weight modifications and the black-hole mechanism are applied sequentially to generate the required IWN. A summary of this more practical version of the integer-weight learning procedure is shown in Table 3.3. A more detailed version of this procedure is available in Appendix B.

How does this new discrete-weight learning scheme compare with the ones discussed in Section 3.5.2? It does not use the direct discretisation approach in which the extra bits are removed through truncation which has been known to result in unacceptable solutions. It does not make the assumption that the acceptable error minima for both the IWN and CWN are located at the same location. It does look for the CWN minima in the beginning, but gradually starts drifting toward an integer-weight solution. The closer it gets to a CWN minimum, the harder it tries to go towards an integer-weight

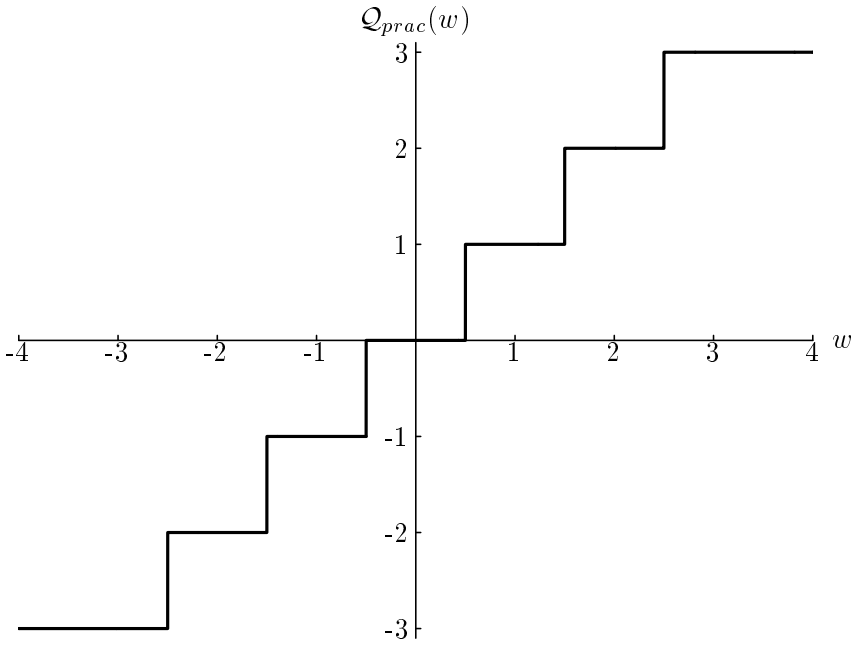


Figure 3.3 $\mathcal{Q}_{prac}(w)$.

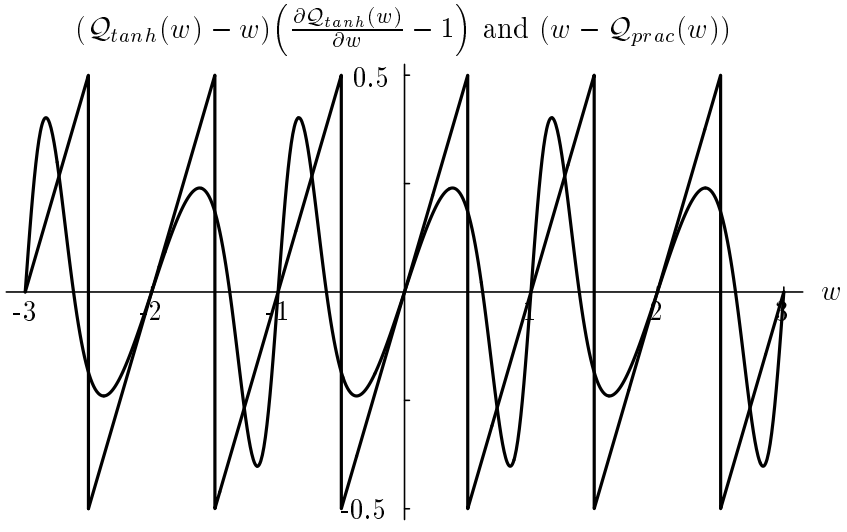


Figure 3.4 Comparison of the actual error term and its approximation in the range $[-3, 3]$.

Table 3.3 Practical IWN learning

Initialise weights in the range $(-0.5, 0.5)$

If $E_w = 0$ and $E_o > \varepsilon$ then boost η by a factor of k_η for just this epoch

Select training example

  Select a weight, w

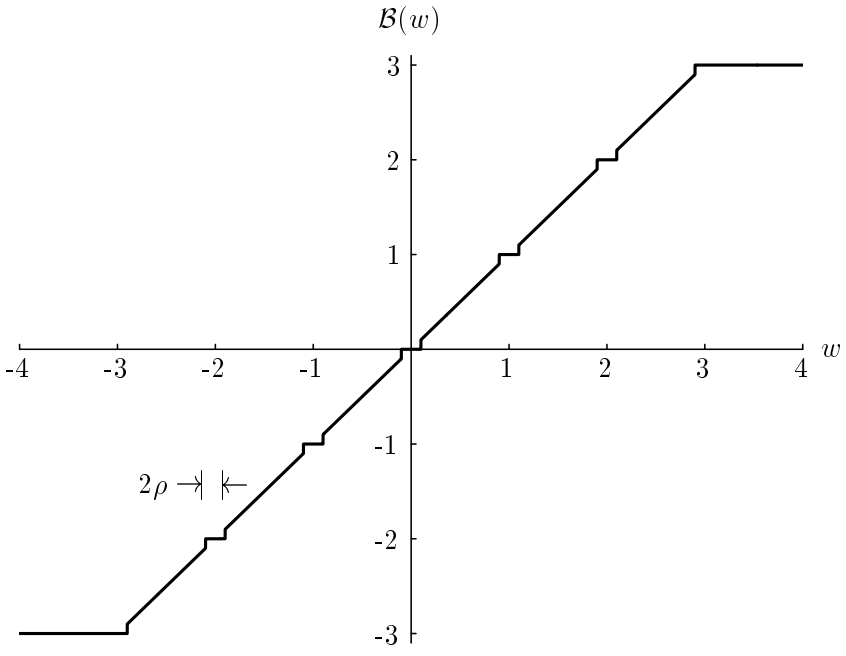
    $w \leftarrow w \; - \; \eta \frac{\partial E_o}{\partial w} \; + \; \mu \Delta w_{o_{old}}$

    $w \leftarrow w \; - \alpha_\chi e^{(\varepsilon - E_o)\beta_\chi} \, (w - \mathcal{Q}_{prac}(w))\tan(RND)$

    $w \leftarrow \mathcal{B}(w)$

  Loop

Loop until $E_o < \varepsilon$ and $E_w = 0$



$$\mathcal{B}(w) = \begin{cases} 3\,\mathrm{sgn}(w) & |w| > 3, \\ \mathcal{Q}_{prac}(w) & |\mathcal{Q}_{prac}(w) - w| < \rho, \\ w & \text{otherwise.} \end{cases}$$

Figure 3.5 The black-hole function.

solution. If E_o starts increasing as the network moves closer to $E_w = 0$, the integer-weight learning procedure changes direction and starts steering the network towards another $E_w = 0$ solution. It uses direct measurements of the error surface's local gradient for calculating weight updates instead of inferring it from indirect measurements. It does not require the expensive recalculation of E_o for every weight adjustment. Its error minimisation mechanism is based on gradient descent, which, on many problems of interest, is quicker than one based on a purely stochastic process. It does, however, have a stochastic mechanism to avoid getting trapped in local minima. Its rounding mechanism, the black hole function, is superficially similar to the probabilistic rounding of Hoehfeld and Fahlman. In fact, the black-hole mechanism is more effective, in that it adapts its strength according to the value of E_o. Moreover, if a weight gets trapped in a black hole, it can only come out due to a very strong push by the BP mechanism.

The main implementation drawback for integer-weight learning is the time consuming calculation of the perturbation term in Equation 3.8, $\tan(RND)$. This term has to be recalculated before the update of each weight. This term is required to perturb the IWN out of the learning paralysis caused by the mutual cancellation of the BP update and the weight-discretisation modification. It was determined experimentally that $\tan(RND)$ worked better than just RND. This may be due to the fact that $\tan(RND)$ has a small value most of the time but infrequently it can become very large (see Figure 3.2). These large but infrequent perturbations can force the IWN out of learning paralysis. Because of its random nature, $\tan(RND)$ can sometimes take the network far away from the E_o minimum, but BP is there to pull the reins and get the network back on course quickly.

The discussion on integer-weight learning was mainly focused on the optimisation of the discretisation mechanism and not on the optimisation of the E_o minimisation mechanism. Only a simple E_o minimisation method, i.e. BP, was considered. This was done to streamline the experiments and to concentrate on the study of interaction of E_w minimisation and E_o minimisation in the simplest possible, but workable, framework. Many E_o minimisation algorithms are available, which although superior to BP in performance on many learning tasks, always require more learning parameters, and

hence complicate the design of learning experiments by a great deal.

There is wealth of literature available on efficient E_o minimisation techniques (see [5] for a survey). Many of these techniques are 2^{nd}-order methods – they use information about the 2^{nd} derivative of the error function $E_o(\mathbf{W})$ to compute weight modifications. The weight changes made by the discretisation mechanism, however, destroy the 2^{nd}-order information at each step, making the superpositioning of the E_o and E_w minimisation mechanisms impractical.

3.6 Functionality Tests

A set of three learning tasks was used for testing the functionality of the integer-weight learning procedure – XOR and two encoder/decoder problems. The XOR problem was selected because of its historical significance and due to the fact that it was the easiest non-trivial learning task. The encoder/decoder problems, 4:2:4 and 8:3:8, were selected because of their closeness to 'real world' pattern classification tasks – small changes in the input pattern cause small changes in the output pattern [35]. A d:q:d encoder/decoder, normally with $q < d$, is an auto-associator, with the number of binary training patterns equal to d, each d-bit pattern consisting of all but one bit 'on' at a time. During the learning phase, the encoder/decoder tries to find an efficient encoding of the input pattern so that it can be squeezed through the hidden-layer bottleneck without any loss of information.

Feedforward networks with offsets and hyperbolic tangent activation function in both the hidden and output layer neurons were used for these simulations. The training data was scaled to the range $[-1, 1]$. These simulations involved clean data therefore the L_∞-norm, E_{o_m}, was used as the error function: training was stopped when E_{o_m} was less than a prespecified ε and all the weights had reached integer values. The values of the discretisation parameters, α_χ, β_χ, α_ρ, and β_ρ, were empirically determined and were found to be problem insensitive. The selected values for these coefficients are shown in Table 3.4.[10] The network was reinitialised if it did not converge to a satisfactory

[10] These same values were used for all of the simulations discussed in this thesis.

Table 3.4 Values of weight discretisation parameters

Parameter	α	β
Weight discretisation rate, χ	0.001	16
Black hole radius, ρ	0.1	6

Table 3.5 Comparison of CWN and IWN Learning epochs

Problem	Config.	Network	Min.	Max.	Avg.	Median
XOR	2:2:1	CWN	9	127	32	22
		IWN	10	77	33	26
Encoder/Decoder-4	4:2:4	CWN	7	31	14	12
		IWN	7	71	20	17
Encoder/Decoder-8	8:3:8	CWN	75	1241	269	172
		IWN	131	808	332	294

solution within a fixed number of epochs, $\boldsymbol{C_R}$.

25 simulations were performed on each the three learning tasks.[11] The results are displayed in Table 3.5. The number of IWN learning epochs is higher compared with the ones for CWN for all three learning tasks. Because of the constraints on its weights, the IWN possesses a limited number of possible functions that fit the learning task, whereas the CWN possesses an infinite number. During the learning phase, the network iteratively checks the suitability of various possible functions on the data to be learned. The CWN reaches an appropriate function very quickly. The IWN, however, has a more arduous route to follow: it has to not only check the fitness of the functions on the learning task at hand but on closeness to the discretisation objective as well. This double checking, combined with the strictness of the second objective, results in longer training times.

Table 3.6 shows the performance results for the direct discretisation of the weights

[11] The training parameters for this and *all* other sets of simulations discussed in this thesis are available in Appendix B.

Table 3.6 Direct discretisation to integer weights of trained CWNs of Table 3.5

Problem	Successful Discretisation
XOR	16%
Encoder/Decoder-4	20%
Encoder/Decoder-8	0%

of the trained CWNs of Table 3.5. The weights were discretised to integer values $\{-3, -2, -1, 0, 1, 2, 3\}$. The results indicate that the easier problems, signified by the smaller number of learning epochs, are more suitable for direct discretisation. The very poor performance of this direct discretisation experiment confirms the conclusions of Chieueh and Goodman [28].

The reason for testing the integer-weight learning procedure on these three simple problems was to check its functionality and to fine tune the learning parameters by monitoring the interaction of the E_o minimisation and the weight-discretisation mechanisms. The fine tuning was made tricky due to the presence of the many adjustable parameters. An exhaustive study of the effect of variations in parameter values was not attempted because of the sheer magnitude of the task. Instead, the values of the weight discretisation parameters were frozen after some initial experiments and only the BP learning rate η was used as the control parameter. The values of μ and ε were also kept constant for these simulations and for those in Chapters 4 & 5 and were varied only for the experiments of Chapter 6.

3.7 Discussion

The integer-weight discretisation scheme proposed in this chapter has a definite hardware implementation advantage. Although the powers-of-two weights are more attractive from the multiplier design point of view, they lack the uniform distribution of simple integers, and therefore have a weaker ability of expression. This new scheme does not contain any problem dependencies. Problem dependent schemes result in more compact solutions but lack the generality of the uniformly distributed integer weights.

The maximum magnitude of the integer-weights was restricted to 3 to make them expressible by 3 bits only. This was done because the initial experiments with 2-bit and 1-bit weights were not successful on the three test problems.

The new integer-weight learning procedure starts off like conventional BP, but becomes more and more discretised in its behaviour as the network gets closer to an error minimum. It does look for the CWN minima in the beginning, but gradually starts drifting toward an integer-weight solution. The closer it gets to a CWN minimum, the harder it tries to go towards an integer-weight solution. If E_o starts increasing as E_w approaches zero, the integer-weight learning procedure changes direction and starts to move towards another $E_w = 0$ solution. Mainly based on steepest descent, it does, however, have a perturbation mechanism to avoid getting trapped in local minima. Its weight rounding mechanism, the black hole function, adapts its strength according to the value of E_o. If a weight gets trapped in a black hole, it can only come out due to a very strong push by the BP mechanism.

This integer-weight learning procedure was found to work well on the three simple learning tasks. These tasks were used to monitor the behaviour and interactions of the various mechanisms present in the learning procedure. Experiments with these tasks were valuable in the fine tuning of the learning parameters. Only the preliminary work with this learning method was presented in this chapter. The more meaningful generalisation-performance results will be discussed in Chapter 6, where a comparison with the performances of CWNs and multiplier-free networks will also be presented.

IWNs require high-resolution arithmetic during the training phase. This makes them unsuitable for in-situ learning. Many commercial neural network applications (for example [135]), however, do not require in-situ training. The number of training epochs for IWNs is generally higher than for CWNs. This is not a major drawback because in many practical applications [159] the learning period occupies only a small fraction of a network's lifetime usage. When products incorporating such applications are mass produced, e.g. a neural network controlled microwave oven,[12] the length of training time becomes a completely insignificant part of the combined design and manufacture

[12]Sharp Logicook microwave oven Model R-4N76.

process.

Another practical issue has to do with input and output variable scaling. In a conventional feedforward network, the scaling of inputs is not compulsory, as the weights can do the scaling automatically during training (although taking more learning epochs compared with the epochs for scaled inputs). In an IWN, however, the weights are range limited, and cannot compensate for the magnitudes of very small but important inputs. The only way they can handle it is through using more hidden neurons, which results in more expensive training. Similarly, for the outputs, if very fine tolerances are set, it may require large weights to achieve those tolerances, which again results in more expensive training [91].

This chapter was exclusively devoted to integer valued weights. The next chapter highlights the weaker learning capability, especially the lack of **affine group invariance**,[13] of this integer-weight scheme, and also discusses the relationship between weight depth and size of the hidden layer.

[13]The property of a group due to which it stays unchanged after the application of a rotation, shift, or scaling transformation.

4

Discrete-weight Approximation of Continuous-weight Networks

4.1 Introduction

The response of a multilayer feedforward network having continuous weights can be approximated with one having discrete weights only. The quality of this approximation can be improved in one of two ways [22] – either by allowing a larger number of discrete levels [40, 169], or by having more hidden neurons [58]. This chapter, while discussing both, will emphasise the latter approach, and will mainly be concerned with weights having small integer values. The experimental results of this chapter suggest that in many cases, the decrease in the quality of approximation caused by finite weight resolution can be offset by an increase in the number of hidden neurons.

Discrete-weight networks have a very attractive feature not found in CWNs – the amount of information stored in their weights is quantifiable [22]. The CWN can store an infinite variety of information, whereas the discrete-weight networks have only a limited capability. Varying the discretisation scheme shows how much complexity is required to get a reasonable approximation to a given learning task. This can be a

significant step towards understanding the fundamental relationship between approximation and memory.

Multilayer feedforward networks with integer weights can be used to approximate the response of their counterparts with continuous-weights. Integer weights, when restricted to a maximum magnitude of 3, require just 3 binary bits for storage, and therefore are very attractive for hardware implementation of these networks. However, these integer-weight networks have a weaker learning capability and lack the affine group invariance of continuous-weight networks. These weaknesses, although compensatable by the addition of hidden neurons, can be used to one's benefit for closely matching the network complexity with that of the learning task.

The purpose of this chapter is to explore the capabilities of discrete-weight networks with the help of the **decision**[1] and error surfaces of a set of 2-D classification problems, which are generalisations of the conventional XOR problem. It is shown that provided a suitable discretisation interval is chosen, a discrete-weight network can be found which performs as well as a CWN, but that it may require more hidden neurons than its continuous-weight counterpart.

The collection of learning heuristics used for the simulations of this chapter is the same as the one used in the last chapter.

4.2 Approximating Continuous-weight Perceptrons

A perceptron with continuous weights has the ability to implement an infinite variety of hyperplanes in its input space. On the other hand, a perceptron having W integer-weights in the range $[-K, K]$, is limited to $(2K+1)^W$ choices only (see Figure 4.1). This restriction is the reason for the lack of the affine group invariance in integer-weight perceptrons. It will be shown experimentally that the addition of a hidden layer to this perceptron can restore this invariance.

All ten data sets shown in Figure 4.2 are linearly separable and therefore can be classified correctly by a continuous-weight perceptron. IWNs with increasing number

[1] A decision surface is a plot of the response of an output neuron with respect to the inputs.

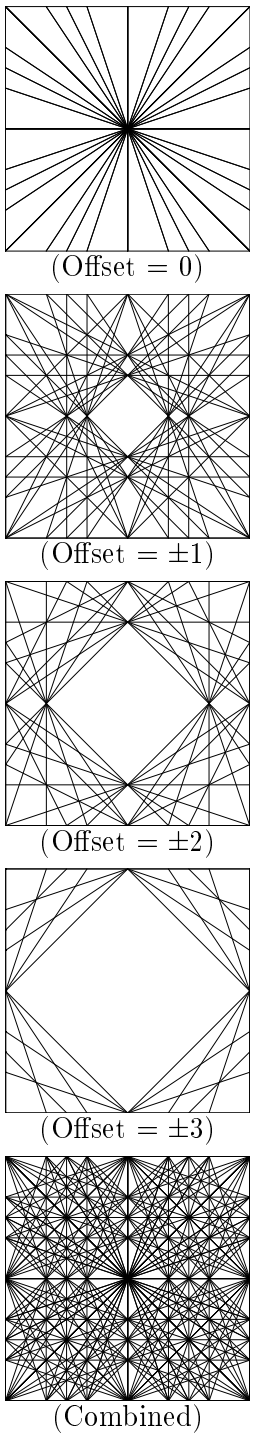


Figure 4.1 The set of decision boundaries of an integer [–3, 3] weight 2-input perceptron with offset. Some of the possible 7^3 decision boundaries lie outside the {(–1, –1), (1, 1)} square, and therefore are not shown.

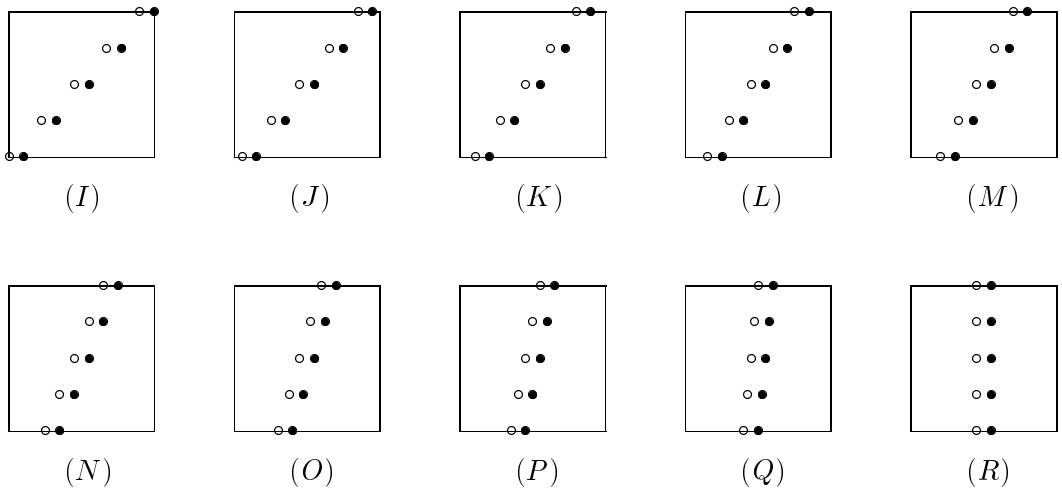


Figure 4.2 Linearly separable data sets with decision boundaries at gradually varying angles.

of hidden neurons were used to classify these data sets and it was found that $E_{o_{rms}}$ decreased (roughly) logarithmically with the number of neurons (see Figure 4.3). This logarithmic rate of decrease in error is different from the theoretically calculated degree of approximation result: E_o was found to decrease at rate of $q^{-1/2}$, where q is the number of hidden neurons [4, 64].

4.3 Approximating CWNs with IWNs

In the work reported here, mappings of the form $f : \mathcal{R}^2 \rightarrow \{-1, 1\}^1$, $\mathcal{R}$ being a closed interval $[-1, 1]$, were used for comparing the IWN and CWN decision surfaces for a set of 10 classification problems (Figure 4.4) which were used for numerous training runs on 2:q:1 networks, with and without **skip-layer synapses**.[2] The results for those simulations are summarised in Table 4.1.

Problems D, D^*, E, and E^* are of the same complexity, as far as the minimum number of required dichotomies is concerned. For all four problems, the CWN requires

[2]Skip-layer synapses are the synapses connecting neurons in two non-adjacent layers. On some problems, their use has been found to result in more compact solutions. Also known as short-cut synapses [78]. Known as main effects in the statistical literature [127].

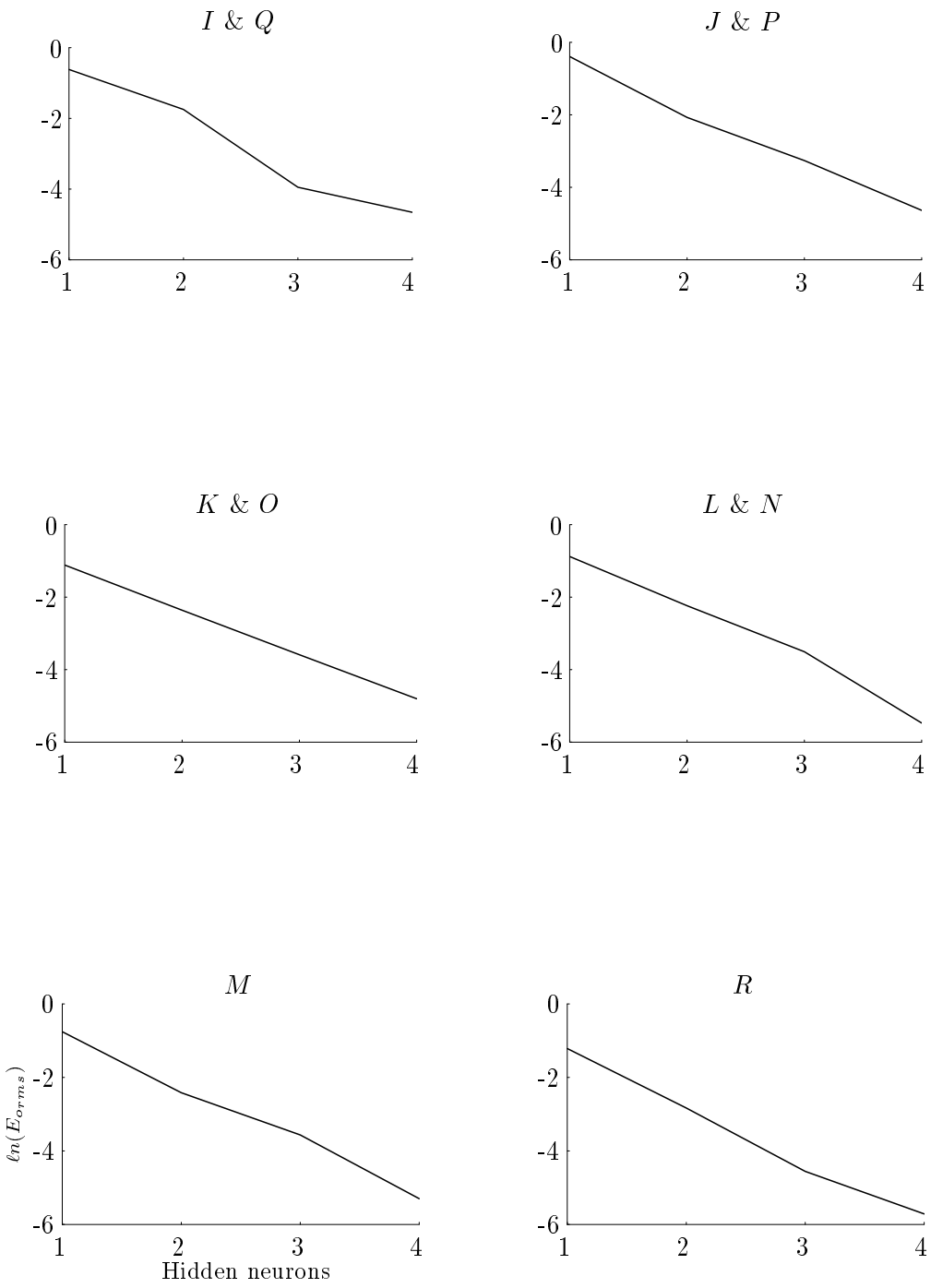


Figure 4.3 IWN minimum $E_{o_{rms}}$ as a function of the number of hidden neurons for the data sets shown in Figure 4.2. The 1 and 2 hidden neuron $E_{o_{rms}}$ were found by exhaustive search, whereas the rest were determined by finding the lowest of at least 10 training runs.

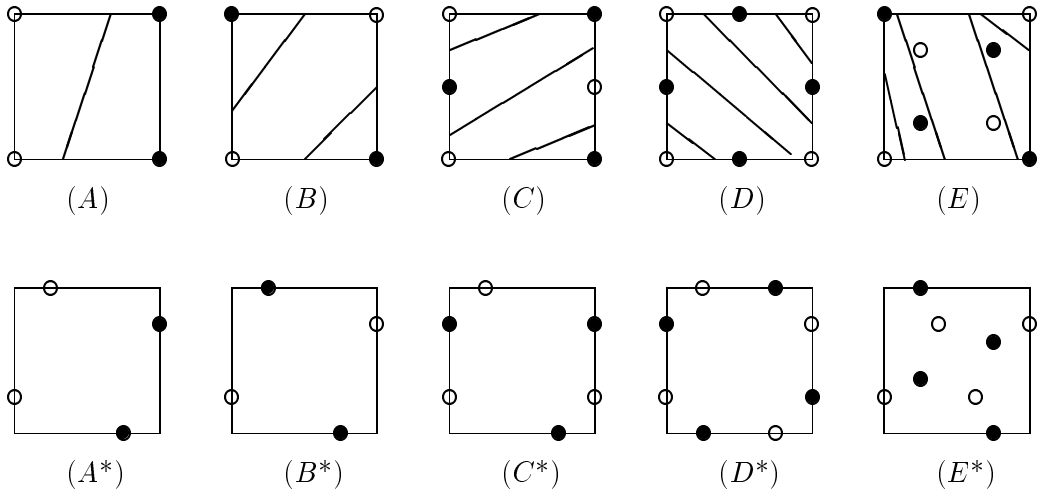


Figure 4.4 Training sets used for comparing the learning capabilities of IWNs and CWNs. The thick lines are the examples of the minimum number of possible dichotomies. *A*–E** are slightly deformed versions of *A–E* with smaller inter-point distances.

two hidden neurons with skip-layer synapses and four without. For the IWN, however, the numbers are somewhat more variable, and larger. This is due to the inability of the IWN to implement dichotomies at arbitrary angles and for arbitrarily close training data points. It should, however, be pointed out that this inability *was* compensated for by the addition of hidden neurons. In a way, this apparent drawback is a beneficial feature of IWNs. This feature provides a fine control on matching the network complexity with that of the learning task.

Problems D and E have the same number of dichotomies, but are different in terms of the angles of the dichotomies. E requires angles in a certain narrow range, whereas D can be implemented with a wider range of angles, and therefore is an easier learning task requiring fewer hidden neurons. A similar comparison can be drawn with respect to D and D^*, and E and E^*, with the added difficulty of smaller inter-point distances in D^* and E^*. This hierarchy of problem complexities is evident in the minimum number of hidden neurons required for the implementation of these problems in IWNs with skip-layer synapses (Table 4.1). In the case of networks without skip-layer synapses, however, this is not the case. This is because these networks have to have at least as

Table 4.1 Comparison of the minimum number of hidden neurons required by CWNs and IWNs for learning problems of increasing complexity. The corresponding decision surfaces are shown in Figures 4.7–4.10 (problems *D*, *D**) and Figures C.1–C.14 (the rest)

		Minimum Number of Hidden Neurons			
		with Skip-layer Synapses		w/o Skip-layer Synapses	
Problem	Dichotomies	$\overrightarrow{\text{CWN}}$	$\overrightarrow{\text{IWN}}$	CWN	IWN
A	1	0	0	–	–
*A**		0	0	–	–
B	2	1	1	2	2
*B**		1	1	2	2
C	3	1	1	3	3
*C**		1	1	3	3
D	4	2	2	4	4
*D**		2	3	4	4
E	4	2	4	4	4
*E**		2	5	4	5

many hidden neurons[3] as the minimum number of required dichotomies in the training data.

A hexagon shaped training set was used to further study the effect of the angle of the training data dichotomies on the learning ability of the IWNs. The hexagon data set of Figure 4.5 was rotated around its centre in 5° steps, and $\overrightarrow{2{:}q{:}1}$ (i.e. 2:*q*:1 network with skip-layer synapses) IWNs were trained on it. The results of those training runs are presented in Table 4.2. Only two data configurations, ones with rotations of 15° and 45°, were learnable with one hidden neuron. These data configurations resulted in exactly the same decision surfaces except for a 90° rotation from the former to the latter. This, once again, shows that IWNs find it difficult to draw dichotomies at certain

[3]This is true in general but not strictly: for example, a 2:2:1 network can implement up to *three* dichotomies in its input space.

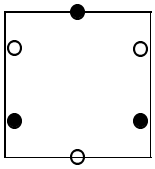


Figure 4.5 Hexagon training set with 0° rotation.

Table 4.2 Comparison of the minimum number of hidden neurons required by an IWN (with skip-layer synapses) for learning the hexagon data set as it was rotated by 5° steps. The decision surfaces for 0° and 15° rotations are shown in Figure 4.6

Rotation	0°	5°	10°	15°	20°	25°	30°	35°	40°	45°	50°	55°	60°
Hidden neurons	2	2	2	1	2	2	2	2	2	1	2	2	2

angles, favouring a discrete set of angles instead.

4.3.1 Repeated Decision Surfaces

Almost all of the integer-weight training runs resulted in some repeated decision surfaces. For example, all skip-layer training runs on the hexagon training set of Figure 4.5 with a 15° rotation resulted in the exactly repeated decision surfaces of Figure 4.6. Similarly, all of the skip-layer runs for problem D resulted in similar decision surfaces, differing only by a rotation of a multiple of 90° (see Figure 4.9 g-l). The reason for the repetition with extra rotation is certainly the 4-fold symmetry of D. The decision surfaces for the skip-layer runs for problem D^* shown in Figure 4.10 have the same similarity except for Subfigure j: this decision surface is an inverted version of the other five. The 'without skip-layer synapses' runs on D and D^* did not result in such regular and exact repetitions: the D^* runs were split into the repetition of two dissimilar patterns, whereas the D runs had at least four distinct patterns.

The repetition of surfaces suggests that the complexity of the network is exactly matched with that of the target problem. Therefore, the problem solution, i.e. the decision surface, can be represented in only one way by the network. The closeness of the match is determined by the complexity of the data and the number of free

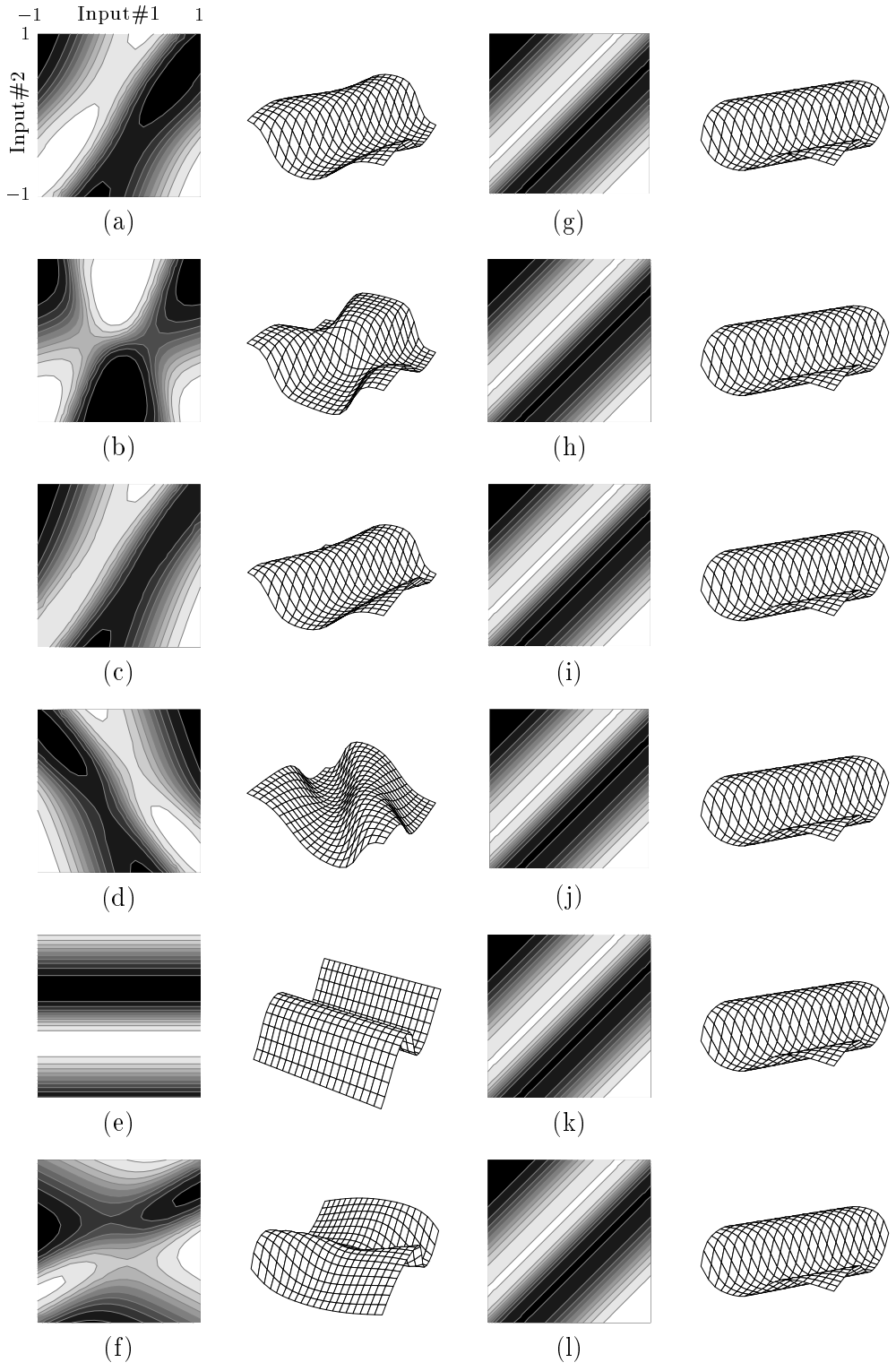


Figure 4.6 Decision surfaces after 6 consecutive training runs on the hexagon data: with a 0° rotation for a $\overrightarrow{2{:}2{:}1}$ IWN (a-f); with a 15° rotation for a $\overrightarrow{2{:}1{:}1}$ IWN (g-l).

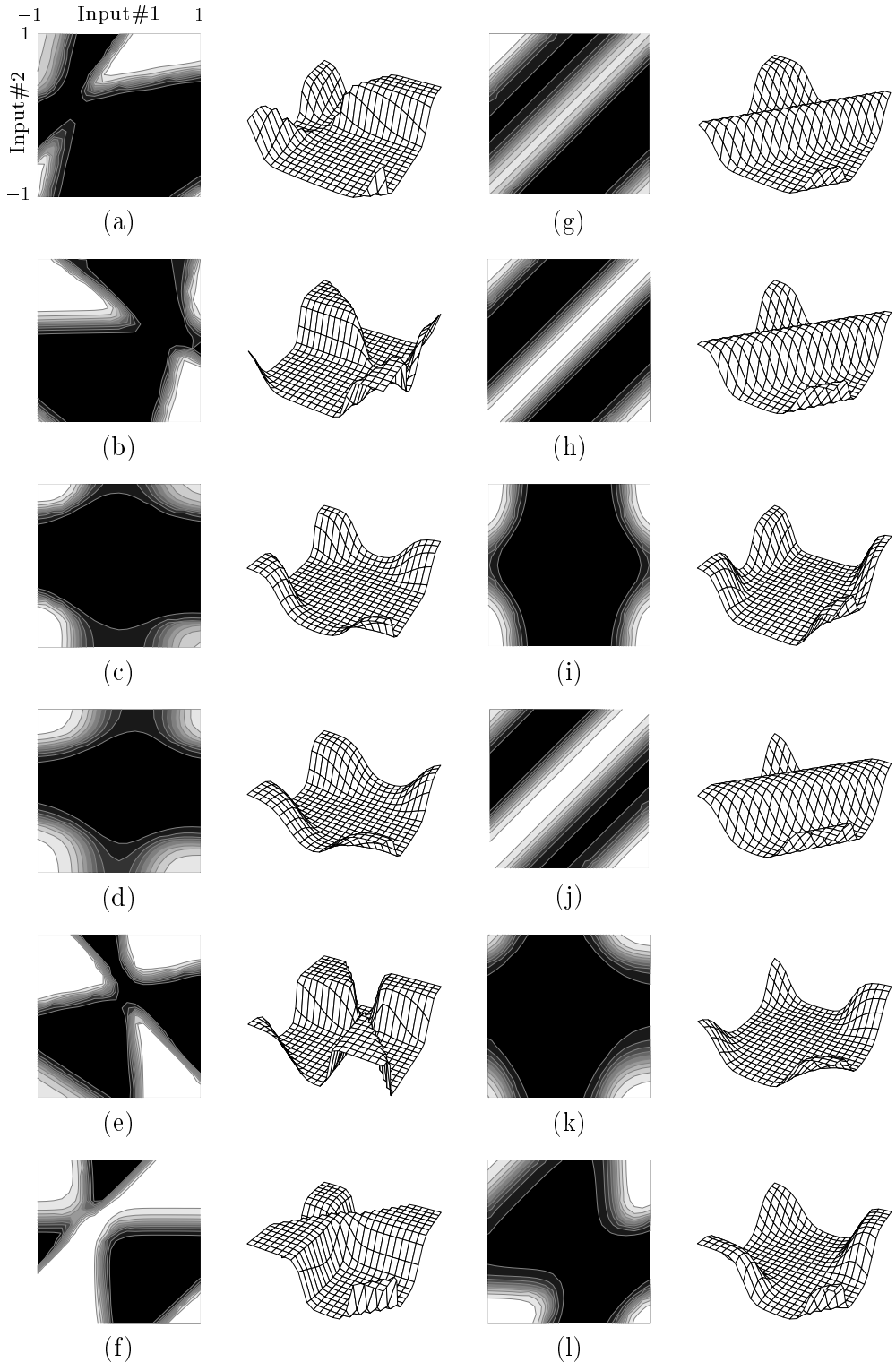


Figure 4.7 Decision surfaces after 6 consecutive training runs on problem *D*: 2:4:1 network with double-precision weights (a-f); 2:4:1 network with integer weights (g-l).

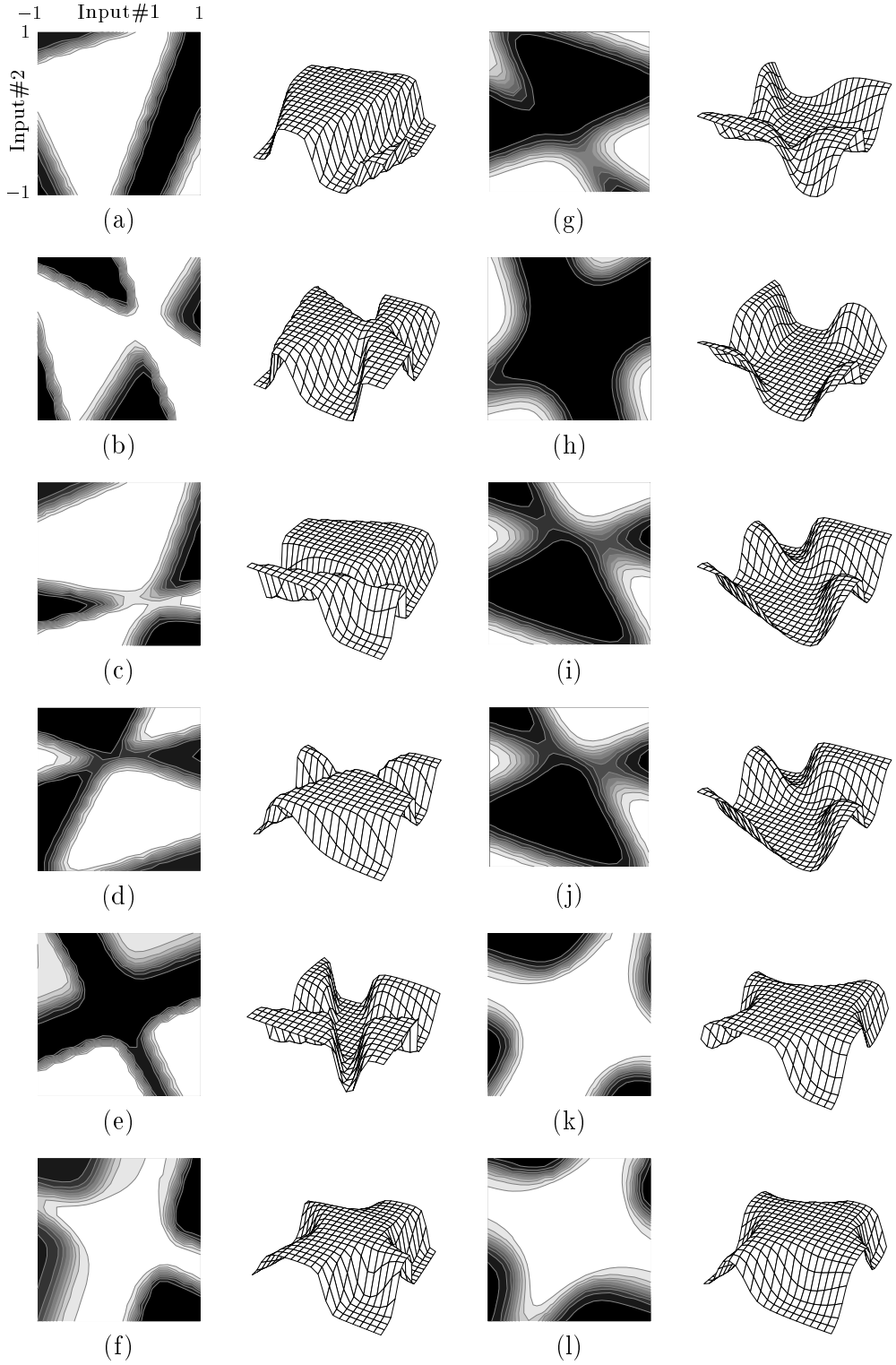


Figure 4.8 Decision surfaces after 6 consecutive training runs on problem D^*: 2:4:1 network with double-precision weights (a-f); 2:4:1 network with integer weights (g-l).

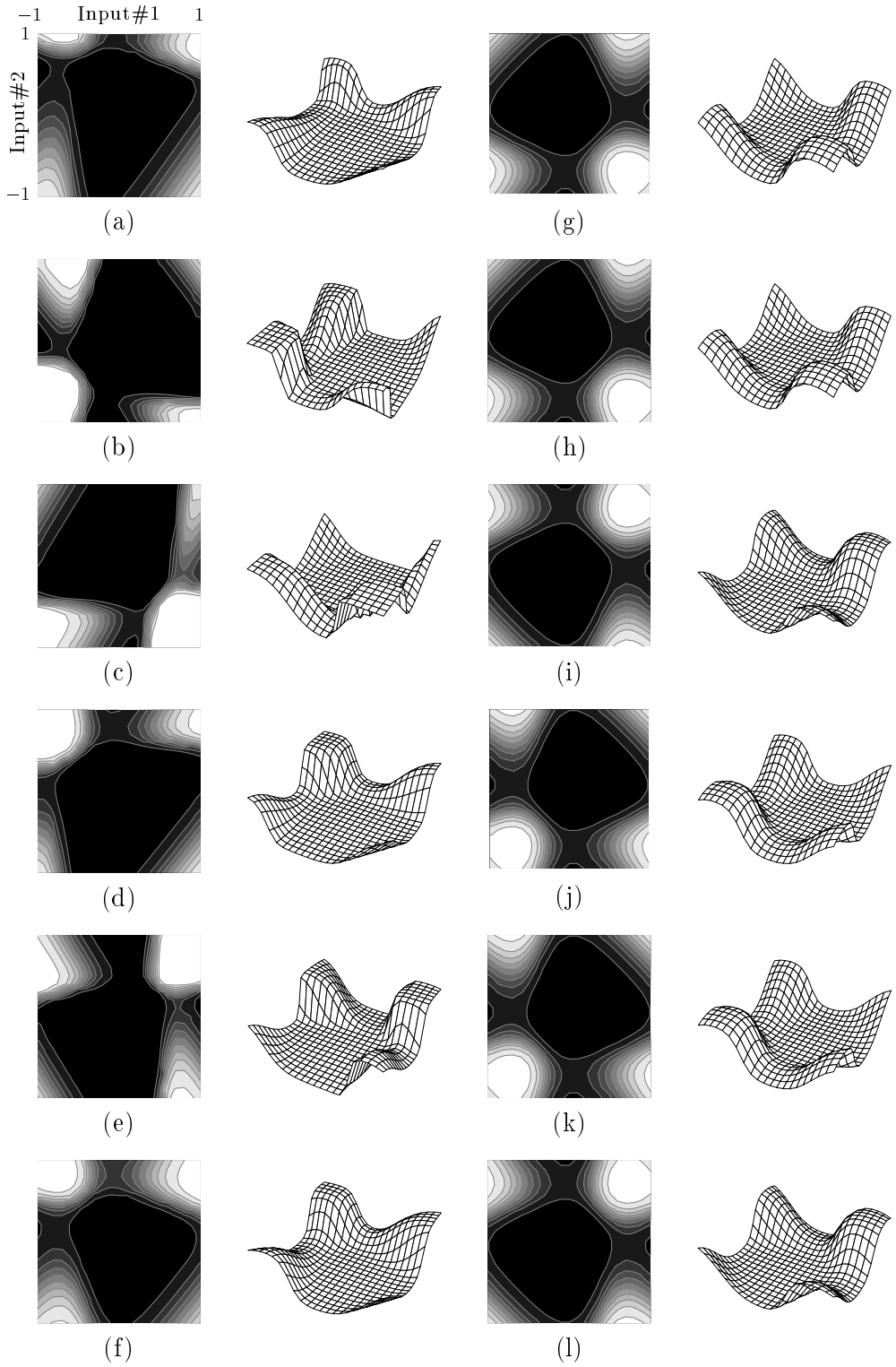


Figure 4.9 Decision surfaces after 6 consecutive training runs on problem *D*: $\overrightarrow{2{:}2{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}2{:}1}$ network with integer weights (g-l).

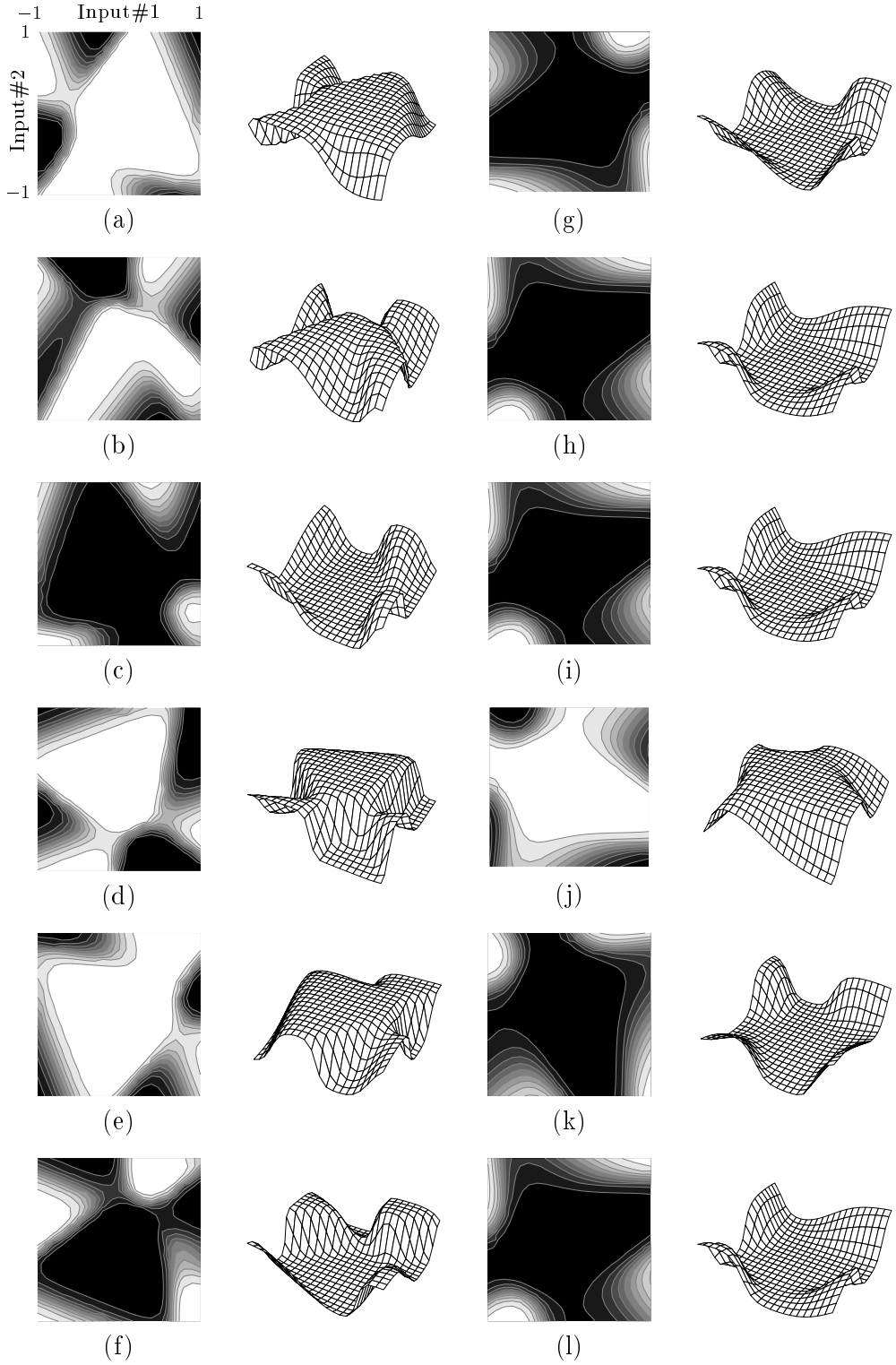


Figure 4.10 Decision surfaces after 6 consecutive training runs on problem D^*: $\overrightarrow{2{:}2{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}3{:}1}$ network with integer weights (g-l).

Table 4.3 Repetition of decision surfaces

Repetitions	Problem	Problem Complexity	Configuration	No. of Weights
Most	D	D	$\overrightarrow{2{:}2{:}1}$	9
2nd most	D^*	$>D$	$\overrightarrow{2{:}3{:}1}$	15
3rd most	D^*	$>D$	2:4:1	17
Least	D	D	2:4:1	17

parameters in the network as is clear from Table 4.3: the problem with the smallest network has the most repetitions, and the simplest problem with the largest network has the least number of repetitions.

Training runs on most of the problems of Figure 4.4 resulted in some duplicated decision surfaces for IWNs, whereas none of the continuous-weight runs resulted in any significant repetitions. The corresponding decision surfaces are shown in Figures 4.7–4.10 (problems D, D^*) and Figures C.1–C.14 (the rest).

4.4 Error Surfaces

Another way of comparing CWNs with their discrete-weight approximations is via their error surfaces. Discrete-weight error surfaces are just the low-resolution sampled versions of the CWN error surfaces, as can be seen in Figure 4.13, where error surfaces for the weight resolutions of 1 and $\frac{1}{8}$ are compared . The locations and values of the global minima for these error surfaces are displayed in Table 4.4. The location of the IWN minimum is the same as that guessed from the location of the one for the weight resolution of $\frac{1}{8}$, although this may not be the case in general. The depths of the two minima are within one half percent of each other. There is, however, an important difference between the two minima: at least one of the training instance was misclassified at the IWN minimum, whereas in the case of the network with $\frac{1}{8}$-resolution, all of the instances were correctly classified at its global minimum.

CWN error surfaces can be analysed to find out whether a network with a certain weight resolution will be able to represent a given input/output mapping. If the CWN

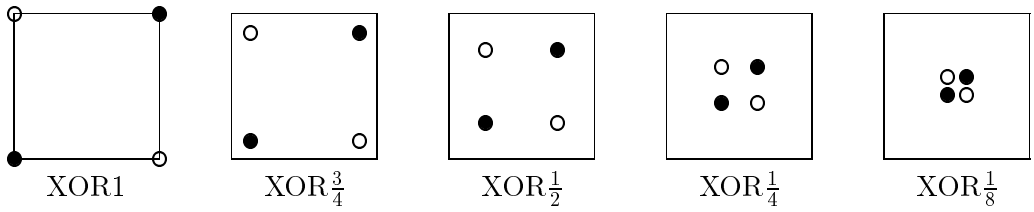


Figure 4.11 XORx training sets.

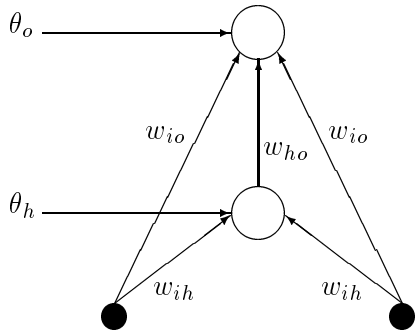


Figure 4.12 Symmetric weight $\overrightarrow{2{:}1{:}1}$ XOR network.

error surface contains sharp minima at points which are missed by the low-resolution sampling, then the discrete-weight network will not be able to represent the same mapping with reasonable accuracy. Figure 4.11 shows the modified versions of the XOR problem used as the test problems to study this phenomenon. These modified versions were produced by gradually bringing the four data points close to each other. As the data points are brought closer, the slope of the error surface in the vicinity of the global error minimum becomes steeper. The minimum looks like the top of a funnel when the points are far apart, as in Figure 4.11 XOR1, and looks more like the middle of a funnel when the points are close by, as in XOR$\frac{1}{8}$. A $\overrightarrow{2{:}1{:}1}$ network with symmetric weights (Figure 4.12) was used to find the depth of the global minima for a number of weight resolutions. Symmetric weights were used to reduce the free parameters of the network from seven to five. It is quite clear from Table 4.5 that, in the case of XOR$\frac{1}{8}$, the error minimum has such a narrow collection region that the IWN cannot 'see' it because of low-resolution weight sampling, and a network with a weight resolution of at least 0.125 is required to correctly classify all the data points, albeit with a relatively

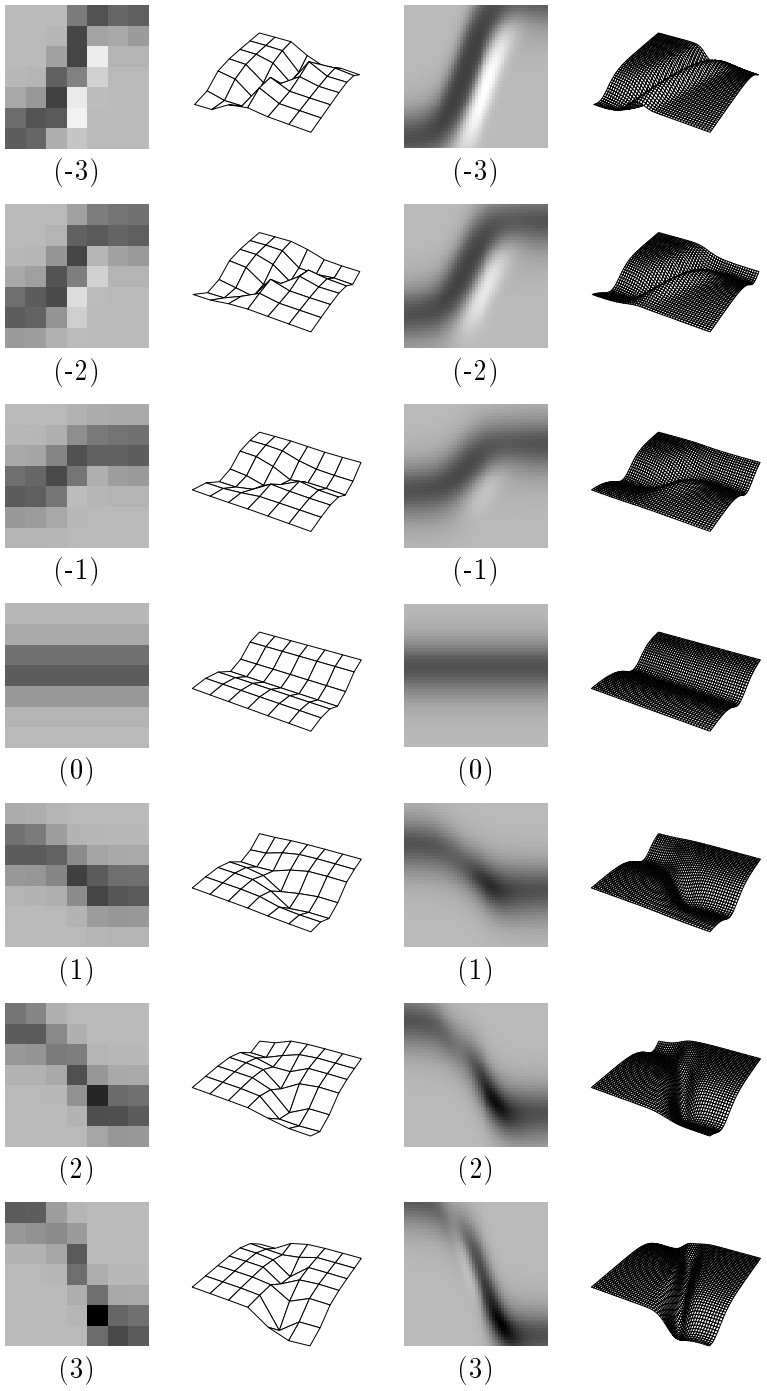


Figure 4.13 Error surface for the network of Figure 4.12 on the XOR$\frac{1}{8}$ problem of Figure 4.11. The 7 images in the left half are for integer weights and the rest are for a weight resolution of 0.125. $w_{ih} = 3$. $w_{io} = -3$. θ_h is plotted along the horizontal axis of the contour plots, and θ_o along the vertical axis. w_{ho} is the figure in the brackets.

Table 4.4 The locations and values of global minima of Figure 4.13. The italicised figure indicates that one or more training instances were *misclassified* at this global minimum

Weight Resolution	w_{ho}	θ_h	θ_o	$E_{o_{rms}}$
$\frac{1}{8}$	3	$1\frac{1}{8}$	$-2\frac{1}{4}$	0.784
Integer	3	1	-2	*0.787*

Table 4.5 Global error minima (each of multiplicity 4) as a function of weight resolution. The italicised figure indicates that one or more training instances were *misclassified* at this global minimum

	Weight Resolution ($w_{min} = -3,\ w_{max} = 3$)								
	6	3	2	1.5	1	0.75	0.5	0.25	0.125
	Number of Bits Required								
	1	2	2	3	3	4	4	5	6
Problem	$E_{o_{rms}}$								
XOR1	*0.879*	*0.875*	*0.501*	0.097	0.210	0.097	0.097	0.097	0.097
XOR$\frac{3}{4}$	*0.722*	*0.722*	*0.719*	0.192	0.177	0.134	0.104	0.104	0.104
XOR$\frac{1}{2}$	*0.875*	*0.875*	0.212	0.148	0.212	0.148	0.148	0.139	0.139
XOR$\frac{1}{4}$	*1.167*	*0.946*	*0.736*	*0.711*	0.540	0.474	0.465	0.446	0.430
XOR$\frac{1}{8}$	*1.088*	*0.991*	*0.977*	*0.958*	*0.787*	*0.859*	*0.787*	*0.787*	0.784

large error.

4.5 Discussion

Decision and error surfaces were used to explore the discrete-weight approximation of CWNs. Although most of the conclusions of this chapter are based on integer-weight results, similar behaviour has been observed for other levels of discretisation.

The CWN can store an infinite quantity of information, whereas the discrete-weight network's storage is finite and depends upon the depth of weights. The latter does,

however, have the advantage of efficient hardware implementation. Moreover, it provides an interesting way of quantifying information stored in a network [22]. Changing the discretisation scheme of the weights of a fixed-size network, or keeping a fixed discretisation scheme while changing the size of the network, are alternative techniques for determining the amount of network complexity required for approximating the response of a CWN to a specified tolerance. These two competing techniques can be used to understand the fundamental relationship between the maximum error admissible in the approximation and the storage capacity required to meet that limit. This extra level of control available on the complexity of discrete-weight networks can be beneficially exploited. Whereas the CWN has only one parameter, the number of hidden neurons, that can be used to adjust its complexity, the discrete-weight network has two: number of hidden neurons and the weight depth. Using the additional selectivity of the weight depth, the network designer can choose a network with a complexity that matches more closely the complexity of the learning task. This will result in improved generalisation performance.

5

Multiplier-free Networks

5.1 Introduction

Feedforward networks with synapses from the set $\{-1, 0, 1\}$ and continuous offsets can be formed without implementing the conventional multiplier in neurons. The reduced complexity of the neurons, combined with the fact that all synapses require a single bit for storage[1] makes these networks very attractive for implementation in hardware.

The number of multiplication operations required for a **forward pass**[2] in a feedforward network is equal to the number of synapses in that network. The quickest way to perform a multiplication in a digital electronic implementation is with a flash multiplier. An n-bit fixed-point VLSI flash multiplier consists of $n \times n$ full adders, each one of which is made up of 31 transistors [2]. If the classic NETtalk network of Sejnowski and Rosenberg [132] was to be constructed in this very fast incarnation, approximately 10^9 transistors will be required for $n = 8$. The slower but more cost-effective option is to have a single multiplier in each hidden and output neuron, and to perform the

[1] A zero valued synapse is not a synapse, it just indicates the absence of one!

[2] Forward pass is the process by which a network computes the output vector in response to an input vector. Also known as recall.

multiplications in a sequence instead of all in parallel. To reduce the transistor count even further at the expense of slower computations, a sequential multiplier can be used. All these cost cutting measures can still not compete with the elegance of the multiplier-free scheme, which, by restricting the synapses to the set $\{-1, 0, 1\}$, eliminates the multipliers altogether, and replaces the expensive and/or slow conventional fixed-point multiplication operation with a fast and simple sign-adjustment instruction. Moreover, the presence of zero valued synapses reduces the time for a forward-pass in digital electronic hardware.

In analogue electronic implementations, the use of Multiplier-Free Networks (MFN) considerably simplifies the manufacture due to the decreased demand for expensive trimming schemes required to achieve the precisely valued synapses required for conventional networks. It should be noted here that the MFN offsets still require full precision trimming. The number of offsets in a network is, however, generally low. For example, the synapse to offset ratio for the NETtalk network is 173 to 1. Moreover, the presence of zero valued synapses also reduces the cost of manufacture of analogue hardware.

In optically implemented networks, where synapse values can be represented as grey-scale masks or voltage levels for spatial light modulator, the complexity of implementation is again reduced because the grey-scale masks and the voltage levels require just three states each – opaque and transparent, and high and low, respectively.

It is possible that the severity of the $\{-1, 0, 1\}$ weight restrictions may weaken the MFN's approximation ability, but simulations on classification tasks indicate otherwise. Comfort is also provided by a new theoretical result on approximation in $C(\mathbb{R})$ presented in this chapter. That result confirms the existence of networks with synapses from the set $\{-1, 1\}$ and unrestricted offsets, that can approximate all continuous functions of one variable to any desired accuracy.

MFNs may have some performance related benefits besides the obvious hardware implementation advantage. For example, the MFN learning procedure generates networks with some weights having a value of zero. The number of these zero-valued weights is higher if larger-than-optimal MFNs are used for training. The reduced num-

ber of effective weights, combined with the fact that all non-zero weights are restricted to only two values, limits the complexity of the network. This can result in reduced over-fitting and therefore improved generalisation performance: it is known from shrinkage estimation and ridge regression analysis in linear models that generalisation can be improved by reducing the size of the weights from estimates that give best fit in the sample [128].

Another potential advantage of the MFN is its relatively immunity to noise in training data. The MFN only captures the main features of the training data in its discrete-valued synapses. Low amplitude noise present in the training data cannot perturb the discrete synapses, because these weights require relatively larger variations in order to jump from one discrete value to another.

The constraints placed upon the network weights may result in an increase in the necessary number of hidden neurons required to achieve a given degree of accuracy on most learning tasks. It should also be noted that the hardware implementation benefits are valid only when the MFN has been trained, as the learning task still requires high-resolution arithmetic. This makes the MFN unsuitable for in-situ learning. Moreover, high-resolution offsets and activation function are required during training and for the trained network.

Previous work on multiplier-free feedforward networks consists of networks having powers-of-two or sums of powers-of-two discrete weights (see for example [77,89]). These schemes were inspired by FIR digital filter design, involving the replacement of the slow and expensive multiplication operation with a series of faster and inexpensive shift and add steps. The network proposed in this chapter is even easier to implement because the multiplication operation has been completely replaced by a single sign-adjustment step.

This chapter starts by presenting the proof of the universal approximation in $C(\mathbb{R})$ property of a multiplier-free feedforward network. The multiplier-free functionality of this SISO[3] network has been made possible by restricting the input layer synaptic strengths to $\{-1, 1\}$. Moreover, the output layer synapses have all been set to a con-

[3]Single-input single-output

stant value equal to 1. It is shown that, in spite of these strong restrictions on synapses, this network can approximate continuous functions of one variable to any desired accuracy. This chapter also provides experimental evidence that MIMO[4] multiplier-free feedforward networks with synapses from $\{-1, 0, 1\}$ and unrestricted offsets are a viable alternative to networks having high-resolution weights. Only the preliminary results on the functionality tests of MIMO multiplier-free networks will be presented in this chapter. The main results on generalisation performance will be held back until the next chapter, where they will be contrasted with those for the conventional continuous-weight networks and integer-weight networks of Chapter 3.

5.2 Universal Approximation

The decision boundaries implementable with integer-weight and multiplier-free perceptrons having two inputs and an offset are shown in Figure 5.1. The very uniform distribution of the decision boundaries, which is controlled by the resolution of the offset, is the distinctive feature of the multiplier-free case. This uniform distribution points towards MFN's possible candidature for being a universal approximator. On the other hand, the integer-weight decision boundaries are not uniformly distributed, supporting the fact that multilayer networks formed with integer weights from $\{-3, -2, -1, 0, 1, 2, 3\}$ lack the universal approximation property (see Section 3.4).

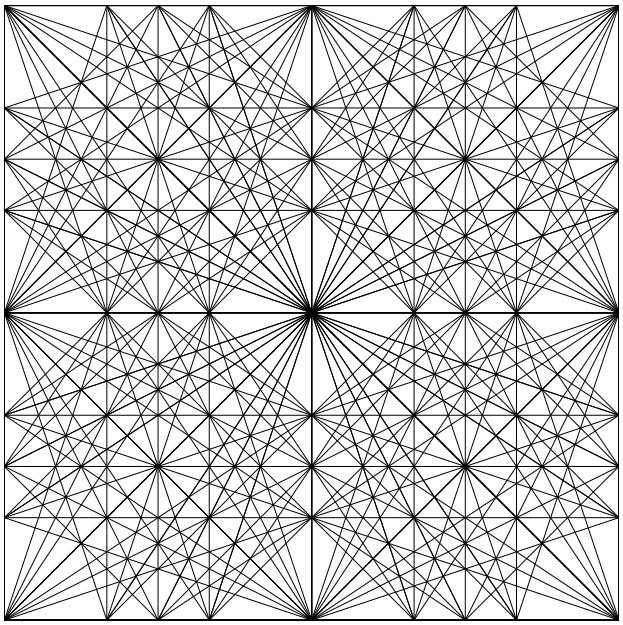
5.2.1 Approximation in $C(\mathbb{R})$

This section establishes that a feedforward network with no output layer synapses, bipolar-binary synapses in the hidden layer, unrestricted offsets, and a single hidden layer of neurons requiring only sign adjustment, addition, and hyperbolic tangent activation functions, can approximate functions in $C(\mathbb{R})$ with arbitrary accuracy.

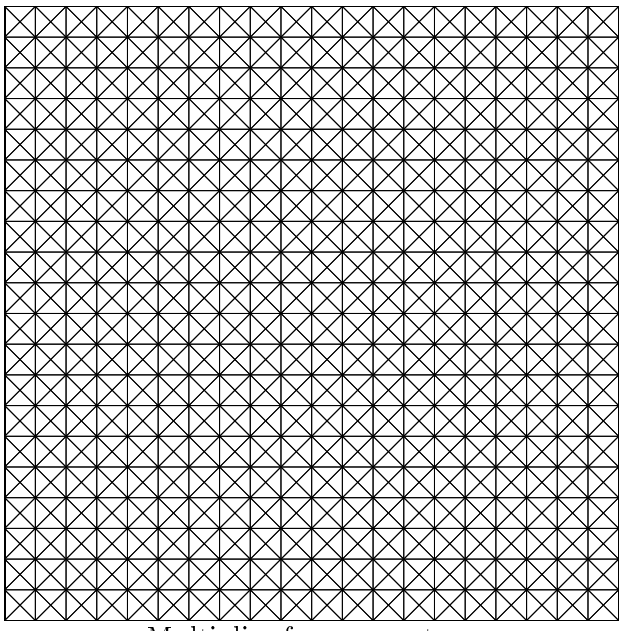
The conventional SISO feedforward network having a single layer of hidden neurons and unrestricted weights can be expressed as:

$$g(x) = \sum_{i=1}^{K} q_i \, \phi(o_i x + p_i), \tag{5.1}$$

[4]Multiple-input multiple-output



Integer-weight perceptron



Multiplier-free perceptron

Figure 5.1 A comparison of the decision boundaries of a integer [–3, 3] weight perceptron (same as Figure 4.1) and a multiplier-free perceptron, each with two input synapses and an offset. For the multiplier-free case the synapses are restricted to the set {–1, 0, 1}, the offset is unconstrained, and this figure shows the boundaries for an offset resolution of 0.1 only. Decision boundaries which lie outside the {(–1, –1), (1, 1)} square are not shown.

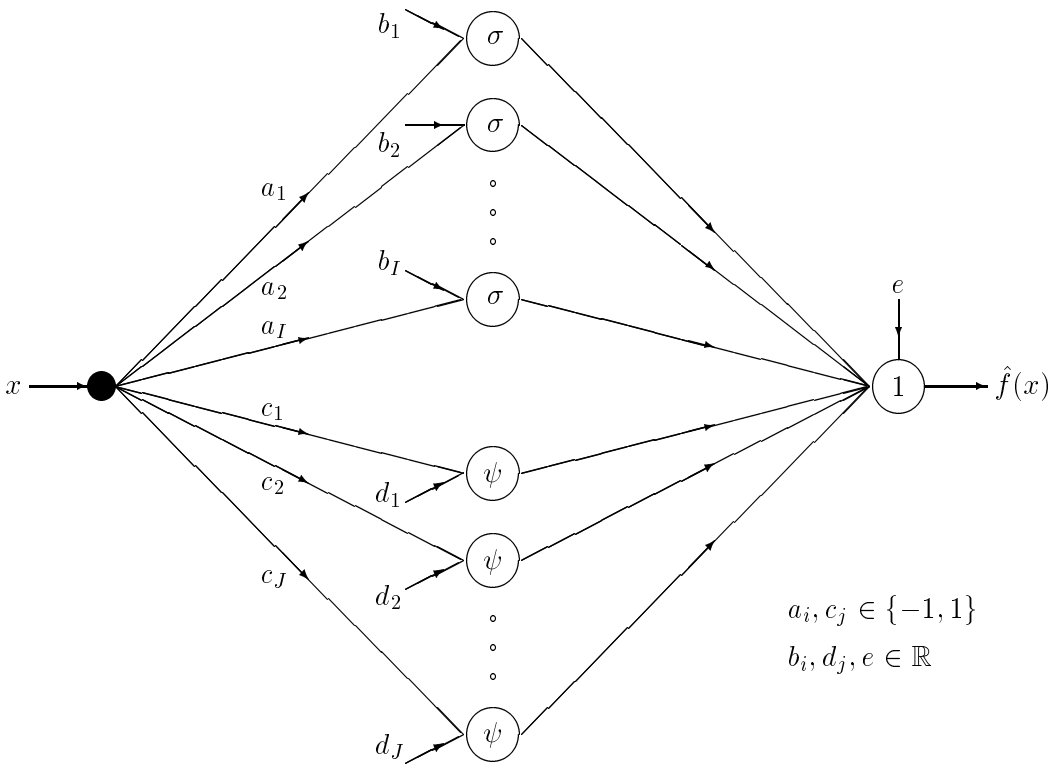


Figure 5.2 Multiplier-free feedforward network.

where ϕ is sigmoidal, and $o_i, p_i, q_i \in \mathbb{R}$. The new multiplier-free counterpart of this conventional network is shown in Figure 5.2. Consider the function $\hat{f}$ of that figure:

$$\hat{f}(x) = \sum_{i=1}^{I} \sigma(a_i x + b_i) + \sum_{j=1}^{J} \psi(c_j x + d_j) + e \tag{5.2}$$

where $a_i, c_j \in \{-1, 1\}$, $b_i, d_j, e \in \mathbb{R}$, $\sigma(\cdot) = \tanh(\cdot)$, $\psi(\cdot) = \alpha\sigma(\cdot)$, and $\alpha \in \mathbb{R}\backslash\mathbb{Q}$. The universal approximation property of $\hat{f}$ can be stated as follows:

Theorem 5.1 (MFN Existence Theorem) *Finite sums of the form $\hat{f}$ are uniformly dense on compacta in $C(\mathbb{R})$.* □

The proof of Theorem 5.1 is dependent on the following two lemmas.

Lemma 5.1 *Let ϕ be a function superanalytic almost everywhere and its derivatives form a basis for continuous functions. Then finite sums of the form*

$$\sum_{i=1}^{K} t_i \, \phi(r_i x + s_i),$$

where $r_i \in \{-1, 0, 1\}$ *and* $s_i, t_i \in \mathbb{R}$, *are uniformly dense on compacta in* $C(\mathbb{R})$. □

Proof This follows immediately from Theorem 2.3. □

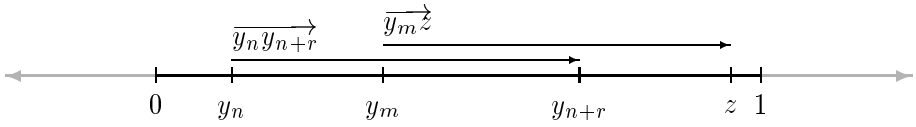
Lemma 5.2 *Given any* $\alpha \in \mathbb{R}\backslash\mathbb{Q}$, *if* $m, n \in \mathbb{Z}$ *then* $m + n\alpha$ *are dense in* $\mathbb{R}$. □

Proof This follows immediately from the following theorem [50]:

Theorem 5.2 (Kronecker's Theorem in One Dimension) *If* $\alpha \in \mathbb{R}\backslash\mathbb{Q}$, $n \in \mathbb{Z}$, *then the set of points formed by the remainders of* $n\alpha$ *is dense in the interval* $(0, 1)$. □

Proof[5] Let $y_n = (n\alpha)$, with $n = 1, 2, 3, \ldots$, where '$(n\alpha)$' denotes the remainder of $n\alpha$. Since α is irrational, y_n is never 0, and no two y_n points coincide. The set of y_n points, S, has therefore a limit point, and there are pairs (y_n, y_{n+r}), with $r > 0$, which can be made near to each other with an arbitrary large value of r.

Let the directed stretch $y_n y_{n+r}$ be a vector. If a stretch $y_m z$, equal to $y_n y_{n+r}$, is marked off in the direction of $\overrightarrow{y_n y_{n+r}}$, from any y_m, then z is another point of S, and in fact y_{m+r}. It should be noted that if the $\overrightarrow{y_m z}$ extends beyond 0 or 1, then the part of it so extending is to be replaced by a congruent part measured from the other end, 1 or 0, of the interval $(0, 1)$.



There are vectors of length less than ε, where $\varepsilon > 0$, and such vectors, with $r > p$ where $p < n$, extending from any point of S, and in particular from y_1. If such a vector is measured repeatedly, starting from y_1, a chain of points,

$$(n_1\alpha),\ (2n_1\alpha),\ (3n_1\alpha),\ \ldots,$$

of any desired length can be obtained. The distance between consecutive points of this chain is less than ε. There is therefore a point $(kn_1\alpha)$ or $(n\alpha)$ within a distance ε of any α of $(0, 1)$. Therefore, $(n\alpha)$ is dense in $(0, 1)$. □

[5] This proof closely follows the approach presented in [49].

Proof of Theorem 5.1 Following the proof of Lemma 2.2[6], it must be shown that $sp(\{\sigma^{(k)}|(-r,r) : k \geq 0\}) = C(\mathbb{R})|(-r,r)$. By the Stone-Weierstrass theorem it is known that for any continuous and strictly monotonic σ the set of polynomials in σ, $\{\sum_k c_k \sigma^k : 0 \leq k \leq j \leq n,\ n \in \mathbb{N}\}$ is uniformly dense in compacta in $C(\mathbb{R})$. Direct calculation shows that $\sigma^{(1)} = (1-\sigma^2)$, $\sigma^{(2)} = -2\sigma(1-\sigma^2) = -2(\sigma - \sigma^3)$, and that the derivative of σ^k is $k \cdot \sigma^{k-1} \cdot (1-\sigma^2) = k \cdot (\sigma^{k-1} - \sigma^{k+1})$. This implies that the set of finite linear combinations of the derivative of σ is exactly the set of polynomials in σ. As $\sigma(\cdot)$ is superanalytic almost everywhere[7], and as its derivatives form a basis for continuous functions, finite sums of the form

$$\sum_{i=1}^{K} t_i\, \sigma(r_i x + s_i), \tag{5.3}$$

where $r_i \in \{-1, 0, 1\}$, $s_i, t_i \in \mathbb{R}$, and $\sigma(\cdot) = \tanh(\cdot)$, are dense on compacta in $C(\mathbb{R})$ by Theorem 2.3. The 0 value of r_i is needed only for providing a constant offset in the output. e can be used for that constant offset and bipolar-binary u_i can be substituted for the ternary r_i.

$$\sum_{i=1}^{K} t_i\, \sigma(u_i x + s_i) + e, \tag{5.4}$$

where $u_i \in \{-1, 1\}$ and $e \in \mathbb{R}$. If t_i is now replaced by $m_i + n_i\alpha$, then according to Lemma 5.2,

$$\sum_{i=1}^{K} (m_i + n_i\alpha)\, \sigma(u_i x + t_i) + e, \tag{5.5}$$

where $m_i, n_i \in \mathbb{Z}$ and $\alpha \in \mathbb{R}\backslash\mathbb{Q}$, is dense in sums of the form of Expression 5.4, and consequently dense on compacta in $C(\mathbb{R})$. Since $m_i, n_i \in \mathbb{Z}$, they can be absorbed in the sums

$$\sum_{i=1}^{I} \sigma(u_i x + t_i) + \sum_{i=1}^{J} \alpha\, \sigma(u_i x + t_i) + e, \tag{5.6}$$

where $I, J \in \mathbb{N}$. It should be noted here that the last step requires $\sigma(\cdot)$ to be a bipolar function to account for negative values of m_i and n_i. Now by substituting $\psi(\cdot)$ for

[6] This lemma is in Chapter 2, its proof, however, is presented in Appendix A, where it is known as Lemma A.4.

[7] $\tanh(\cdot)$ is superanalytic everywhere except at 0.

$\alpha\sigma(\cdot)$, and making some cosmetic changes, the final result is achieved:

$$\sum_{i=1}^{I} \sigma(a_i x + b_i) + \sum_{j=1}^{J} \psi(c_j x + d_j) + e,$$

which is same as the $\hat{f}$ of Equation 5.2 and now has been shown to be dense on compacta in $C(\mathbb{R})$. Theorem 5.1 is proved. □

The MFN existence theorem, Theorem 5.1, guarantees that an arbitrary continuous function of one variable can be approximated to any desired degree of accuracy with the feedforward network of Figure 5.2. This network trades off the complexity of individual neurons with a possible increase in their number. Learning algorithms similar to the one proposed in Chapter 3 can be used to train networks of this type. Although the complexity of the learning algorithm is increased by the presence of two distinct activation functions in the hidden layer, it is some what compensated for by the complete lack of weights in the output layer.

The constraints placed upon the network weights may result in an increase in the necessary number of hidden neurons required to achieve a given degree of accuracy on most learning tasks. In addition, the hidden layer is not homogeneous in the proposed model, and requires two types of neurons – differing, however, in their activation functions only. Conventionally, these activation functions have been implemented in hardware with look-up tables. If the MFN is implemented as a distributed processing system then each hidden neuron will have its own look-up table. In that case, the requirement of two types of hidden neurons will not result in any additional hardware-complexity.

5.3 MIMO Multiplier-free Networks

The universal approximation result that was proved in the last section was for a SISO network.[8] No theoretical evidence for universal approximation in $C(\mathbb{R}^d)$, i.e. universal approximation of multivariable continuous functions, with similar networks is currently

[8]This SISO result implies universal approximation for the SIMO (single-input multiple-output) case as any SIMO network can be constructed as an ensemble of SISO networks.

available, although the author conjectures it to be true. This conjecture was tested with a series of experiments with MIMO MFNs. The MIMO MFNs used for these experiments differed with the SISO MFNs of the last section in two ways: first, they were allowed to have output layer weights, and these weights were restricted to $\{-1, 0, 1\}$ to preserve the multiplier-free functionality of the network; secondly, the α of Figure 5.2 was set equal to 1, which resulted in a homogeneous hidden layer. Neither step was necessary for the success of these experiments, but they were taken to improve the size and speed of the implementation: the first step makes unnecessary the redundant presence of two hidden neurons with outputs differing in polarity only and allows the flexibility of not connecting some hidden neurons to some of the output neurons; the second step makes unnecessary the presence of two separate types of hidden neurons, without noticeably degrading the learning performance during the experiments.

5.4 Multiplier-free Learning

The collection of heuristics used for the MIMO MFN simulations is a modified version of the one described in Chapter 3 – the modifications being that the synapses are now restricted to $\{-1, 0, 1\}$ instead of $\{-3, -2, -1, 0, 1, 2, 3\}$ and that the offsets are not discretised and are modified solely according to the conventional BP procedure.

It is worthwhile to note here that, like IWNs, MFNs are more suitable for learning classification tasks than function approximation tasks. This conclusion can be drawn by examining the decision boundaries of the 2-input multiplier-free perceptron (Figure 5.1). These boundaries have only four possible slopes, and all other slopes required for a given learning task have to be formed using a combination of these four. This, in general, will result in an impractically large number of hidden neurons to meet the finer error tolerance requirements of function approximation tasks.

5.5 Functionality Tests

The functionality of MFN learning was checked by performing 25 simulations each on the XOR, and 4:2:4 and 8:3:8 encoder/decoder problems. The setup for these

Table 5.1 Comparison of CWN and MFN Learning epochs

Problem	Config.	Network	Min.	Max.	Avg.	Median
XOR	2:2:1	CWN	9	127	32	22
	2:3:1	MFN	53	2778	792	467
Encoder/Decoder-4	4:2:4	CWN	7	31	14	12
		MFN	17	116	39	33
Encoder/Decoder-8	8:3:8	CWN	75	1241	269	172
		MFN	90	544	199	133

simulations was the same as the one used in Chapter 3. The results are shown in Table 5.1.

The training runs on the XOR problem with a 2:2:1 MFN were not successful – the E_{o_m} did not meet the ε requirement of 0.4 after numerous runs. The minimum achievable E_{o_m} was 0.42. The inability of the 2:2:1 network to learn the XOR problem can be explained in terms of the maximum allowed strengths for the synapses. This maximum strength and the slope of the linear region in the middle of the neuron activation functions (see Figure 1.2) are interchangeable quantities. The steepness of the slope determines the sharpness of the decision surface close to the decision boundary. This sharpness, in turn, determines the error in the decision close to the decision boundary. Correct decisions that are located close to the decision boundary result in small errors if the slope of the activation function is large. For CWNs and IWNs, the larger allowed strengths of synapses result in sharper effective slopes for activation functions, which in turn results in smaller possible E_o. In the MFN case, however, the maximum magnitude of synapses is limited to one. It is clear from Figure 5.1 that the MFN *does* possess the appropriate dichotomies in its repertoire that can separate the training points of the XOR problem successfully. The decision surfaces around those dichotomies are, however, not steep enough to result in an acceptable E_{o_m}. In the 2:3:1 case, the total input to the output neuron is larger because of the increased fan-in, and therefore, the network is able to create sharp enough decision surfaces around the

dichotomy that result in low enough E_{o_m}. The number of learning epochs is, however, still large. This is due to the small number of possible solutions present in the weight space. Any further increase in the number of hidden neurons will certainly result in a smaller number of epochs.

5.6 Discussion

The new feedforward network paradigm proposed in this chapter lends itself to very efficient hardware implementation because it does not require a conventional multiplier for its operation – the expensive and/or slow multiplication operation has been simplified to a single sign-adjustment instruction. This was accomplished by restricting all synaptic values to $\{-1, 0, 1\}$. This synapse scheme has the added benefit of storage efficiency.

The multiplier-free networks are, in general, expected to be larger than their conventional counterparts in terms of the number of hidden neurons, but should be more compact in hardware due to the absence of the conventional multipliers. The total number of multiplication operations required for a forward pass is equal to the number of non-zero synapses, $S_{\neq 0}$, in each network. Experimental results of this and the next chapter show that CWN and MFN require similar number of non-zero synapses to achieve similar performances. To achieve the fastest forward pass, the CWN requires $O(n^2)$ transistors to implement each n-bit fixed-point flash multiplication operation, whereas the MFN requires $O(1)$ transistors for a single 1-bit conditional sign-changer. Therefore, it can be concluded that for similar execution speeds, the cost of implementing the multiplication operations in a CWN is $O(n^2 S_{\neq 0})$ compared with $O(S_{\neq 0})$ for the MFN.

An MFN existence theorem was also presented in this chapter. This theorem guarantees that a feedforward network with no output layer synapses, bipolar-binary synapses in the hidden layer, unrestricted offsets, and a single hidden layer of neurons requiring only sign adjustment, addition, and hyperbolic tangent activation functions, can approximate all functions of one variable with any desired accuracy. A multivari-

able extension on this theorem is currently not available, but the experimental evidence of this chapter points toward its existence.

MFNs require high-resolution arithmetic during the training phase. This makes them unsuitable for in-situ learning. Many commercial feedforward network applications (for example [135]), however, do not require in-situ training. The possibly large number of learning epochs required for MFNs as compared with CWNs should not be a deterrent to their use as, in many practical applications [159], the learning period occupies only a small fraction of a network's lifetime usage. When products incorporating such applications are mass produced, e.g. a neural network controlled microwave oven, the length of training time becomes a completely insignificant part of the combined design and manufacture process.

In general, discrete weights are expected to result in larger networks. Because of their larger size, the complexity of the discrete-weight networks is more granular than that of CWNs. This finer granularity can be exploited to select a network with a complexity that matches more closely with the complexity of the learning task. This should result in improved generalisation performance. A comparison of the generalisation performance of MFNs with that of CWNs and IWNs will be presented in the next chapter. These results will show that the MFNs are as effective generalisers as conventional CWNs.

6

Generalisation Experiments

6.1 Generalisation Performance

This chapter is concerned with comparing the generalisation performance of the two new paradigms proposed in this thesis, IWN and MFN, with that of the conventional continuous weight feedforward network. Although ease of implementation and learning time are important factors which influence the choice of a feedforward network paradigm, generalisation performance – the accuracy of a trained network on **unseen data**[1] [15] – is the key metric which determines a paradigm's usefulness.

A trained network will be a good generaliser if it has learned the concept embedded in the training data [16]. Training a network which will be a good generaliser is not a trivial task because most real-life applications demand training with noisy data. A good generaliser will have a smooth input to output mapping, which generally means that it will not have many large weights [16]. Also, its complexity will match that of the 'noise-free' version of the mapping embedded in the training data.

The weights of IWNs and synapses of MFNs are limited in their magnitudes. They also usually have larger hidden layers, compared with their CWN counterparts. Because

[1] Unseen data is the data that was not used to train that network.

of the larger number of hidden neurons in their hidden layers, their complexity is more finely selectable than that of CWNs. This finer granularity can be exploited to select a network with a complexity that matches more closely with the complexity of the learning task at hand. It was found in Chapters 3 & 5 that some of the weights of the trained IWNs and MFNs have zero values. This reduces the number of superfluous parameters in the model. These three factors – smaller weights, a more finely granular complexity, and zero weights – point towards a simpler model, which should result in IWNs and MFNs having a generalisation performance better than or equal to that of the CWNs.

The methodology of the generalisation experiments presented here was to train many CWNs, IWNs, and MFNs on a given set of data in similar circumstances, to pick as representative of each of the three paradigms the network which was the best generaliser, and compare the performances of those representatives. Three data sets, exemplifying three very different of fields of feedforward network applications, were used for these experiments. All of these sets – the MONK's benchmark, the Pima Indian diabetes database, and the handwritten numeral recognition database – are publicly available and have been used in the past by many other researchers for the benchmarking purpose [9, 21, 25, 31, 44, 47, 48, 93, 113, 134, 143, 145, 146, 149, 150, 162].

This chapter provides experimental evidence that IWNs and MFNs, despite having severely constrained weights, are as effective generalisers as conventional feedforward networks. It starts by discussing the techniques that are commonly used for estimating the generalisation performance. Various regularisation techniques used for tailoring the complexity of a network with respect to the learning task at hand are then presented. The target network of the generalisation experiments, the optimal network, is then defined. After presenting the methodology of the learning experiments, this chapter concludes with the results on the three data sets.

6.2 Estimation of Generalisation Performance

Moody [97] has proposed the following relationship for estimating the error on test data, $E_{o_{ts}}$, from the sum-of-squares error on T training examples, $E_{o_{tr}}(T)$ [16]:

$$E_{o_{ts}} = \frac{2E_{o_{tr}}(T)}{T} + 2\sigma^2\frac{W_{\mathit{eff}}}{T} \tag{6.1}$$

where W_{eff} is the number of effective weights in the CWN, and σ^2 the variance of the noise on the training examples. The number of effective weights is usually smaller than the total number of weights in the CWN [97] and has to be estimated. The effective number of training examples may also differ from the actual. σ is another parameter that needs to be estimated. The difficulty of accurately calculating these three estimates makes the prediction of $E_{o_{ts}}$ using Equation 6.1 infeasible. Useful conclusions can however be drawn from the form of this relationship. For achieving the best generalisation, i.e. the lowest $E_{o_{ts}}$ to $E_{o_{tr}}$ ratio:

1. The number of training examples should be maximised.
2. The variance of noise on training examples should be minimised.
3. The number of effective weights, or the total number of (non-zero) weights in general, should be minimised.

The third conclusion points towards the better suitability of the IWN and MFN as generalisers as compared with the CWN.

6.2.1 Empirical Estimation

Comparative experiments for the empirical estimation of generalisation performance can be set up in one of three ways [95, 153]:

1. **Train-and-test** A random sample containing one half of the total number of examples is selected. This subset is used to train the network, while the remaining examples are used to test the network once it has been trained. The performance of the trained network on the test set is an estimate of the generalisation performance.

2. **Cross-validation** The data set is divided equally into k randomly selected, mutually exclusive subsets called folds. $k-1$ networks are trained sequentially on all combinations of $k-1$ folds, while the performance of the trained networks is tested on the one remaining fold. The average of $k-1$ such errors is an estimate of the generalisation performance metric.

3. **Bootstrap** A random sample is selected by **sampling with replacement**[2] from the data set and is used to train the network. The trained network is then tested on the remaining data. This procedure is repeated a large number of times. The average of all such test errors is an estimate of the generalisation performance metric.

The train-and-test technique is recommended for large (>1000 samples) data sets, cross-validation for ones with intermediate sizes, and bootstrap for small data sets [95]. For classification learning tasks, the size of data sets should be measured by the number of examples of the smallest classification group, which is termed as the **effective sample size**. The generalisation performances estimated by the three techniques converge as the size of the data sets grow. In terms of implementation efficiency, train-and test is the fastest of the three, bootstrap the slowest. The latter two techniques are quite useful for comparing the relative performances of a given set of network paradigms but they fail to deliver a single network representing the paradigm that will be the best generaliser.[3] The train-and-test technique, however, does produce such a network – the one having the best performance on the test data.

It should be noted here that the test data subset mentioned in the train-and-test experimental setup for estimating the generalisation performance is not used in any way during training, and that this test data is not the same test subset as is used for

[2]Sampling with replacements may result in successive samples being not mutually exclusive, some of the examples may never appear in any of the samples, and there may be repetitions within an individual sample.

[3]The outputs of the $k-1$ cross-validation networks can however be combined with the help of a second level 'master' network, trained with respect to the desired outputs of the training data, to form a large single network [163]. This technique is called 'stacked generalisation'.

'optimal stopping of training' to be discussed in the next section.

6.3 Regularisation Techniques

Regularisation techniques avoid over-fitting and automate the model selection process. They result in smoother models, which in turn, results in better generalisation performance. Regularisation is achieved by enforcing smoothing and complexity minimising constraints on the model so that the resultant is the least complex model which fits a given set of training data [16]. Common regularisation methods are:

1. **Weight decay** is the technique in which a penalty term is added to the training cost function which penalises large weights. A common penalty term is the sum of the square of weights [57]. This technique is known as penalised ridging in the statistical literature.

2. **Weight elimination** is the technique in which the number of non-zero weights is the penalty term [51, 81, 110, 152]. The integer-weight learning procedure presented in Chapter 3 incorporates an implicit weight elimination mechanism.

3. **Weight sharing** is the technique in which the number of independent weights is the penalty term [105, 122].

4. **Addition of noise to weights** during training results in robust internal representations which have better generalisation performances [99, 100].

5. **Addition of noise to inputs** during training yields a smoother function by providing small variations to the input [73, 131]. This technique is known as smoothed ridging in the statistical literature.

6. **Optimal stopping of training** uses two sets of data during training, one for training and the other for testing, to stop the training when error on the test data hits a minimum. It has been successfully tested in many experiments where the number of examples is relatively large compared with the number of adjustable parameters in the network [103, 130, 151].

7. **Bayesian methods** define a set of model regularisation parameters, e.g. the strength of the weight decay mechanism, and make the estimation of the optimal values of those parameters a part of the learning process. This is the most recent, and potentially the most useful, technique for training feedforward networks. The 'exact' implementation of this technique suffers from the slow speed of training, the recent 'approximate' implementations, however, hold much promise [87, 130, 141].

The regularisation method selected for the experiments presented here is 'optimal stopping of training'. This popular technique is fast, works well with larger-than-necessary networks, and does not require the introduction of any new training parameters except the ratio of train/test data split [130]. The CWN experiments presented here use weight decay in addition to optimal stopping of training. Weight decay was used to stop weights from reaching excessively large values. This makes the comparison between IWN and MFN, and CWNs more meaningful, because IWNs and MFNs are not allowed to have large weight values. Avoiding large weights assures that the network learns smooth mappings [16].

6.4 Optimal Network

The definition of the optimal network used for the experiments presented here is based on the conjecture called **Ockham's Razor**[4] [141]:

> *If two different models have similar performances on a set of data then the* simpler *of the two should be preferred.*

The optimal configuration is therefore defined to be the network configuration with the smallest number of hidden neurons which results in the lowest number of misclassifications on the test data set. It should be noted that for CWNs, the test data performance may not degrade if the number of hidden neurons is increased beyond the optimal number provided that the 'optimal stopping of training' method is used [130]. There is,

[4]This conjecture, also known as Occam's Razor, about economy in explanations, was systematically applied, but never explicitly stated by the 13th century philosopher, William of Ockham [141].

however, a problem with using larger than optimal networks – although smaller networks are not guaranteed to be better in general, Baum and Haussler have shown in [7] that for a given training set and training error, the worst-case error bounds on unseen data vary proportionally with the number of weights in a network. Moreover, larger-than-optimal networks are slower and demand more storage space. It will be shown experimentally bellow that, for IWNs and MFNs, if larger than the necessary number of hidden neurons are used then the number of zero weights increases automatically to reduce the effective complexity of the network.

6.5 Generalisation Experiments

All three data sets used for the experiments presented here are publicly available and have been used by many other researchers for benchmarking. They represent three very different of fields of feedforward network applications. The first one, MONK's benchmark, is an artificial set and was designed to compare the capabilities of a number of learning algorithms, consists of three separate classification tasks, one of which is noisy, and each of which has binary-coded inputs and a single binary output [143, 146, 162]. The second set, onset of diabetes prediction data set, has both discrete and continuous inputs, may have some irrelevant inputs, may have much noise in the inputs, may have a high degree of correlation between inputs, and has a single binary output [9, 25, 93, 113, 134, 145, 149, 150]. The final set, handwritten numeral recognition data set, is a representative of the image processing applications, has a high degree of information redundancy, has discrete inputs, and many binary outputs [21,31,44,47,48].

6.5.1 Methodology

The *only* goal of the experiments presented in chapter is to compare the generalisation capability of the three paradigms, CWN, IWN, and MFN, in similar circumstances. Keeping this in mind, the classification error probability on the test data was used as

the metric of comparison, not **sensitivity**[5] or **specificity**[6] as the goal is to compare learning capability of three paradigms and not the usefulness of the model.

The train-and-test technique was chosen to estimate the generalisation performance because of its simplicity of implementation. Only two, instead of the recommended three, data subsets were, however, used for this purpose – the first one for training, and the second one for optimal stopping of training as well as estimating the generalisation performance. The dual use of the second subset is not strictly appropriate as it has been used during training and therefore the performance metric extracted from it is not an unbiased estimate of the generalisation performance of the trained network [125]. In the case of the experiments discussed here, however, the emphasis is on comparing the generalisation ability of the three paradigms in similar circumstances, and not estimating their *true* generalisation performance. This particular emphasis on comparison was the main reason for the use of just two data subsets. Moreover, all of the previous results cited here on MONK's benchmark and the handwritten numeral recognition database have used the same strategy [31, 47, 143, 146].

The train-and-test method has another use in addition to the optimal stopping of training – network size or complexity selection. Figure 6.1 shows how a network can can be evaluated on the test data after every training epoch for optimal stopping of training. If training is continued any further, there is a risk that the network will become **over-trained** and therefore over-fit the training data, resulting in poor generalisation performance. The second use of train-and-test is shown in Figure 6.2. Here, networks of various complexities, i.e. sizes in terms of the number of hidden neurons, are evaluated on the test data after training has been completed to select the network with the best test data performance. For CWNs, however, this test data performance may not degrade if the network size is increased beyond the optimal provided that the train-and-test method is used for optimal stopping of training [130].

[5]Sensitivity of a decision is the likelihood that an event will be detected if it occurs. It is the ratio of true positives to the sum of true positives and false negatives. This metric is especially of importance when it is critical that an event be detected. Also known as True Positive Ratio [34].

[6]Specificity of a decision is the likelihood that the absence of an event is detected given that it is present. It is the ratio of the true negatives to all negatives. Also known as True Negative Ratio [34].

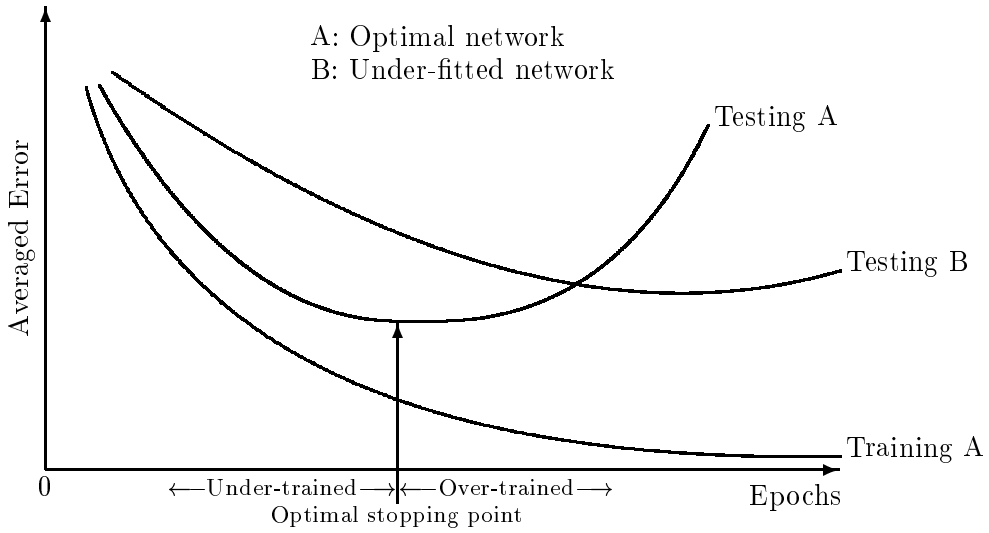


Figure 6.1 Using train and test-every-epoch for optimal stopping of training.

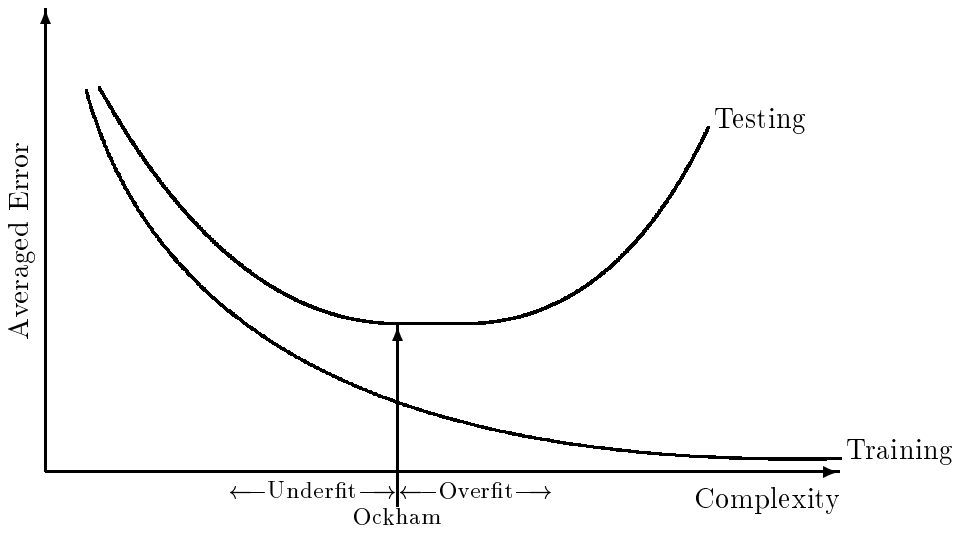


Figure 6.2 Using train and test-at-the-end to select network size or complexity.

The IWN and MFN were tested after they had converged to a discrete solution. If the performance on the test data was not satisfactory then the BP learning rate η was temporarily boosted by a factor of k_η for one epoch to push them out of the local minimum and training was continued until the next discrete solution. If the network did not converge in C_R epochs, the training procedure was reinitialised without reinitialising the training epochs counter, $\boldsymbol{C_T}$. Network configurations with zero hidden neurons were tested first, and then hidden neurons were added until the network performance on test data failed to improve. At least a hundred training runs were performed for each network configuration, and many hundreds for promising ones. Test data results reported here represent the best performance of the optimal configurations.

6.5.2 MONK's Benchmark

The MONK's benchmark[7] was proposed by Wnek et al. [160–162] and has been used for comparing the generalisation performances of 25 learning techniques by the creators or advocates of those techniques [143, 146]. These techniques included both machine learning and feedforward network paradigms. The feedforward network paradigms were conventional BP, BP with weight decay, Cascade-correlation [143], and **Alopex**[8] [146]. This benchmark is based upon an artificial robot domain. Each robot in this domain can be described by specifying values for a set of six attributes. Each one of these six discrete attributes can have one of 3, 3, 2, 3, 4, and 2 values, respectively, which results in 432 possible combinations, which constitutes the total data set. The learning benchmark consists of three binary classification problems. Each of these problems is concerned with determining the membership of a robot to a particular class. The goal is to learn the logical class description from only a subset of the description of the 432 robots that can be classified.

[7] This benchmark is publicly available in the directory `ftp://ics.uci.edu/pub/machine-learning-databases/monks-problems/` [98]

[8] Alopex is a stochastic learning procedure which uses local correlations between changes in individual weights and changes in the cost function to update those weights. This procedure predates the current resurgence in neural network learning by more than a decade and was originally proposed for mapping visual receptive fields [146].

Table 6.1 Network configurations for the MONK's benchmark. The Alopex simulations used a slightly different input encoding resulting in 15 inputs only

	CWN trained with					
Problem	BP [142]	BP with Weight Decay [142]	Cascade Correlation [36]	Alopex [146]	IWN	MFN
1	17:3:1	17:2:1	17:1:1	15:3:1	17:4:1	17:3:1
2	17:2:1	17:2:1	17:1:1	15:3:1	17:3:1	17:2:1
3	17:4:1	17:2:1	17:3:1	15:3:1	17:1	17:2:1

Problem 1 (Attribute 1 = Attribute 2) OR (Attribute 5 = 1)

This problem is in standard Disjunctive Normal Form (DNF). 124 examples were selected randomly from the data set for training, while the remaining 308 were used for testing.

Problem 2 Only two attributes = 1

This problem is similar to the parity problem and is difficult to describe in DNF or Conjunctive Normal Form. 169 examples were selected randomly from the data set for training, while the rest were used for testing.

Problem 3 (Attribute 5 = 3 AND Attribute 4 = 1) OR
(Attribute 5 $\neq$ 4 AND Attribute 2 $\neq$ 3) with added noise

This problem is also DNF but with 5% deliberate misclassifications in the training data set which consists of 122 examples. The remaining 310 examples were used for testing.

For meaningful comparisons, the simulations discussed here use the same training/test data subsets as were used in [143, 146]. Each possible value of every attribute was assigned a single bipolar-binary input, resulting in a total of 17 inputs. The L_∞-norm, E_{o_m}, was used as the error function for the first two problems as they are noise-free, whereas the L_2-norm, $E_{o_{rms}}$, was used for the third one as it contains some misclassifications. IWNs and MFNs with various 17:q:1 configurations were trained and the configurations resulting in the best generalisation performance are shown in

Table 6.2 Comparison of generalisation performance on the MONK's benchmark

	CWN trained with					
Problem	BP [142]	BP with Weight Decay [142]	Cascade Correlation [36]	Alopex [146]	IWN	MFN
1	100%	100%	100%	100%	100%	100%
2	100%	100%	100%	100%	100%	100%
3	93.1%	97.2%	97.2%	100%	100%	100%

Table 6.1. A comparison of the generalisation performance of the IWN, MFN, and CWNs on the MONK's benchmark is presented in Table 6.2. It is clear from this table that IWNs and MFNs are at least as capable generalisers as the best generated by continuous-weight learning algorithms.

It is very interesting to note that the IWN solution for problem 3 does not involve any hidden neurons, suggesting that the problem is **linearly separable**[9] – which in fact is the case if the deliberate misclassifications are removed from the training data [36]. This result points towards the robustness of IWNs for handling noise in the training data. The IWN only captures the main features of the training data in its discrete-valued weights. The small number of noisy examples present in the training data cannot perturb the discrete weights, because these weights require relatively large variations in the input data in order to jump from one discrete value to another. The networks having continuous synapses or offsets, and trained by steepest descent techniques, are *obliged* to vary their weights by very tiny amounts so that the average error in the outputs is minimised. This results in their failure to capture the concept embedded in the training data, and fitting to the noise instead.

It was stated in Chapters 3 & 5 that the proposed integer-weight learning procedure generates IWNs and MFNs with many weights having a value of zero and that the number of zero-weights is higher if larger-than-optimal networks are used for training.

[9]Linear separability is the property of a classification task by which the members of every class can be separated from the ones from all other classes by a single hyperplane.

Table 6.3 Zero valued weights in the IWNs and MFNs trained on the MONK's benchmark

		Configuration		Weights	
Network	Problem	Startup	After Training	Zero	Effective
IWN	1	17:4:1	no change	52	25
	2	17:4:1	17:3:1	20	38
	3	17:1	no change	12	6
MFN	1	17:3:1	no change	24	34
	2	17:3:1	17:2:1	2	37
	3	17:7:1	17:2:1	23	16

This statement is well supported by these simulations on the MONK's benchmark as can be seen in Table 6.3. Averaging over the three problems, 55% of the IWN weights have a values of zero. The equivalent figure for MFNs is 36%. These figures are for effective network configurations obtained after training, and not for configurations at the start of training, as some of the hidden neurons had zero valued synaptic connections with the output neuron, and therefore did not have an active role in the final configuration.

6.5.3 Forecasting the Onset of Diabetes Mellitus

This data set[10] is concerned with a group of adult women belonging to the Pima Indian tribe and was collected by the U.S. National Institute of Diabetes and Digestive and Kidney Diseases [8,71,72]. The learning task is to forecast the onset of diabetes mellitus according to the World Health Organisation criterion[11] within five years of a clinical examination. A set of eight risk factors, which were recorded during that clinical examination, are used to make this prediction. 768 such clinical histories constitute the data set out of which 268 are for patients who tested positive for diabetes within five years of the clinical examination and the rest were found to be healthy.

[10]The data set is publicly available in the directory `ftp://ics.uci.edu/pub/machine-learning--databases/pima-indians-diabetes/` [98]

[11]The patient has diabetes mellitus if the plasma glucose concentration after 2 hours in an oral glucose tolerance test is 200mg/dl.

49% of the 768 clinical records had zero values for attributes which cannot be zero. These are most probably missing values [114]. Moreover, it should be noted that this database may be very noisy: some of the attributes may be unimportant. Some attributes may have been measured incorrectly, some of the most important attributes may have been completely missing, and one of the attributes, the diabetes pedigree function, is based on the heuristic combination of many pieces of information. The complete list of the input variables is as follows:

1. Number of times pregnant.
2. Plasma glucose concentration after 2 hours in an oral glucose tolerance test. This is the standard test for diagnosing diabetes.
3. Diastolic blood pressure (mm Hg). An above normal value is a risk factor.
4. Triceps skin fold thickness (mm). It indicates the degree of obesity.
5. 2-hour serum insulin (mu U/ml). It is below normal for insulin dependent diabetics but not for non-insulin dependent diabetics, and therefore, is not a good predictor [101]. This was the attributes missing in most of the cases with missing attribute values.
6. Body mass index (kg/m^2). It also indicates the degree of obesity.
7. Diabetes pedigree function. It represents the hereditary risk factor.
8. Age (years).

From a clinical point of view, the glucose concentration, diabetes pedigree, age, and obesity are the major risk factors, the insulin level is of some importance, and the blood pressure and number of pregnancies are the least important indicators [101]. It will be interesting to compare this ranking with the weight structure of the trained networks.

This database has been analysed with the help of connectionist tools in [9, 25, 93, 113, 134, 145, 149, 150]. Smith et al. used the ADAP algorithm, a neural-like algorithm which uses Hebbian learning to build associative models, to learn from a randomly

selected subset of 576 cases, and have reported a sensitivity and specificity of 76% on the test set consisting of the remaining 192 cases. This equality of the two figures was achieved by thresholding the continuous 0–1 output of the model at 0.448 [134]. Michie et al. used a set of 22 machine learning, neural, and statistical techniques to analyse this data [96]. They used 12-fold cross-validation to determine generalisation performance. They achieved their best results of 87.7% with **logistic discriminant analysis**,[12] whereas a figure of 75.2% was obtained with a CWN trained using BP. Wahaba et al. [150] used 500 cases for training and 252 for testing with a **smoothing spline model**,[13] and report a generalisation performance of 76%. This study deleted 16 cases from the database because those cases had some attributes with impossible values. Bioch et al. [9] used the same data configuration and a **Bayesian classifier**[14] which resulted in a generalisation performance of 88.7%. Carpenter and Markuzon [25] have also analysed this database and have reported a generalisation performance of 77% with ***k*-nearest neighbours**,[15] 78% with **ART-EMAP**,[16] and 81% with **ARTMAP-IC**.[17] Ripley [115] used 200 cases for training and 332 for testing, and ignored the rest because of missing values. The study reports that the best CWN results were obtained without a hidden layer of neurons. The best overall results of this study was a generalisation performance of 81% obtained by using a **mixture representation**[18]

[12]Logistic discriminant analysis chooses classification hyperplanes with respect to maximising a conditional likelihood cost-function and not optimising a quadratic cost function which is the case for linear discriminant analysis [93].

[13]Smoothing spline modelling is piecewise approximation by polynomials of degree n with the requirement that the derivatives of the polynomials are continuous up to degree $n-1$ at the junctions [20].

[14]Bayesian classifier assigns a class to an object in such a way that the expectation value of misclassification is minimised. Also known as the minimum risk classifier, and belief network.

[15]k-nearest neighbours is a clustering algorithm that minimises the the sum of squares of distances between the training data and k points.

[16]ARTMAP with added spatial and temporal evidence accumulation processes. **ARTMAP** is a supervised learning procedure explicitly based on neurobiology.

[17]ARTMAP with an instance counting procedure and a new match tracking algorithm.

[18]Mixture representations of data use a linear combination of Gaussian distributions to represent arbitrary distributions [10].

Table 6.4 Comparison of generalisation performance on the forecasting diabetes database

Network	Configuration	Zero Weights	Effective Weights	Gen. Perform.
CWN	8:2:1	–	21	78.4%
IWN	8:6:1	26	35	76.9%
MFN	8:3:1	6	25	78.0%

and the **EM algorithm**[19] [116].

For the simulation results presented here, a **balanced data set**, i.e. a set having equal number of positive and negative cases, of 536 cases was randomly selected. This set was then split into two equal, balanced subsets for training and testing. All eight attributes were then standardised to zero mean and unit variance [13].

$$x \longleftarrow \frac{x - \mu_x}{\sigma_x}$$

where μ_x is the calculated mean, and σ_x the standard deviation. All attribute values outside the $[-1, 1]$ range were truncated to $\{-1, 1\}$. The standardisation of inputs to zero-means and small magnitudes along with initialising the network with small uniformly distributed weights has the advantage that most of hidden neurons start with their outputs in the linear region of the activation functions, where the learning progresses the fastest [129]. Moreover, Le Cun et al. have analytically shown that standardising the inputs to zero mean improves the convergence properties of BP learning [82].

A comparison of the generalisation performance of the best IWN, MFN, and CWN on the diabetes database is presented in Table 6.4. The three generalisation performances are within 1.5% of each other, with CWN being the best and IWN the worst. One expects IWN to be the worst performer, but why is the MFN, with its smooth and simple mappings due to the presence of weights with small magnitudes and many zero weights, not performing better or equal to the CWN? Is it because the universal approximation conjecture about its capabilities is wrong? That is a possible reason.

[19] Expectation Maximisation (EM) algorithm calculates the probability density of observations based on parameters and not observations.

Another possible reason may have to do with a fault on account of the experimental procedure: it is possible that the two hyperplanes drawn by the hidden neurons of the CWN are positioned in such a way that it requires the superpositioning of a large number of MFN hidden neurons to emulate them. The MFN experiments were performed with a maximum of 19 hidden neurons and the best solution that was found used only a fraction of those 19 as that was the best that can be found with up to 19 hidden neurons. It is possible that with more hidden neurons a better solution can be found but the price of that minute increase in accuracy is an impractically large hidden layer.

Keeping in mind the discussion about the noise-immunity characteristic of the IWN on the MONK's problem 3, a valid question to ask here is that why did the IWN not outperform the other two paradigms on this diabetes data set? This data set, just like MONK's problem 3, is noisy and therefore should result in better performance with the IWN. The answer to this question may lie in the amount of noise present in the data – MONK's 3 had only 5% corrupted examples, whereas in the present case the number, although unknown, is likely to be higher, as is indicated by the large number of misclassification on both training and testing data.

As an aside, it was found that all three paradigms agreed in rating the glucose concentration test and age to be the most important attributes. This first-order approximate rating was determined by summing of the absolute values of synapses connecting the inputs to the hidden neurons.

6.5.4 Handwritten Numeral Recognition

The numeral database[20] used for these simulations was collected at Bell Labs [21, 31, 44, 47, 48]. It consists of 1200 isolated handwritten samples of numerals 0..9. Twelve individuals were asked to provide 100 samples each while following a given writing style, resulting in 120 examples of each numeral. The sample style given to the writers was similar to the one required for the U.S. Internal Revenue Service's machine readable tax form 1040EZ [1] (see Figure 6.3). The raw images of those samples were normalised

[20] This database has been kindly made publicly available by Isabelle Guyon at `ftp://hope.caltech.edu/pub/mackay/data/att.database`

Figure 6.3 Sample style for writing numerals on the machine readable U.S. Internal Revenue Service tax form 1040EZ [1].

and then thresholded to fit a 16×16 binary pixel grid. This database was then carved into two subsets of 600 samples each – the first five samples of each numeral from every writer were used for training and the rest for testing (see Figures 6.5 & 6.6). Guyon et al. [47], and Druker and Le Cun [31], both have reported a generalisation performance of 97% with a 256:20:10 network trained using conventional BP, and a 256:40:10 network trained using 'double backpropagation', respectively. For the simulations presented here, the 256 element matrices were transformed into 32 element vectors by summing the rows and columns as shown in Figure 6.4. Each element of the these 32-element vectors was then transformed as

$$x \longleftarrow \frac{x - 8}{8}$$

The distributions of the resultant input features are shown in Figure 6.7 for all ten numerals, for both training and testing data. It should be noted that in that figure, to the human eye, each of the ten numerals is clearly distinguishable from all others even after this $256 \rightarrow 32$ inputs transform. Although this 8-fold reduction in input dimension did cause a 4% reduction in generalisation performance (compared with [31, 47]), it made the running of many more simulations possible due to the reduced memory and CPU requirements. The numeral classification decisions were taken according to the neuron with the maximum signal in the output layer.

CWNs, IWNs, and MFNs with various 32:q:10 configurations were trained and the results for the best performing networks are shown in Table 6.5. The lack of the universal approximation property in a IWN may be the cause of the slightly lower performance of the IWN, whereas the finer control over network complexity may have caused the slightly better performance of the MFN. It should, however, be stressed that the differences between the generalisation performance of the networks representing the three paradigms are not significant enough to claim the superiority of one over any of

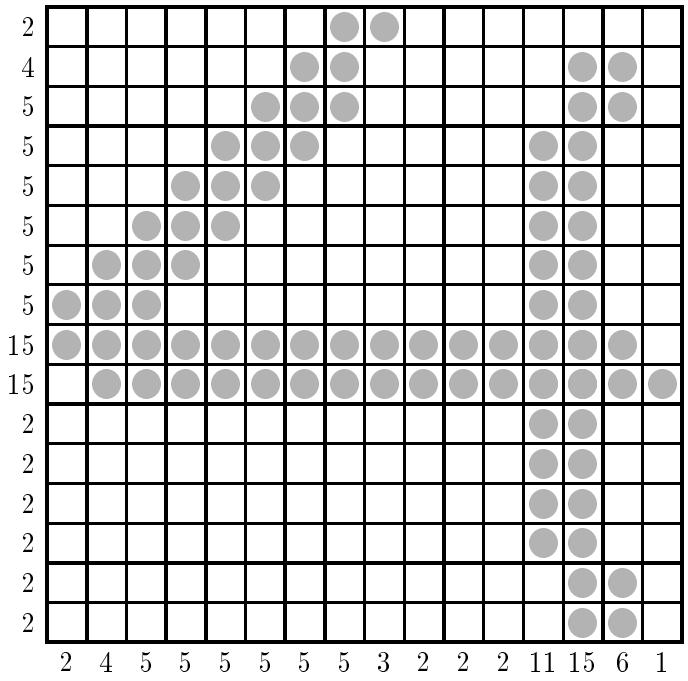


Figure 6.4 Feature reduction (256 → 32) of the handwritten numeral data. The 32 row and column sums instead of the 256 pixel values were used for both training and testing.

the rest. Nevertheless, it can be concluded from these results that the very strong constraints placed on the weights of IWNs and MFNs have not significantly hampered their performance.

Table 6.6 shows the details of the errors made by the three networks of Table 6.5. The number of misclassification on each numeral are similar for the CWN, IWN, and MFN for all save one numeral: '3'. In this case, the CWN made 10 mistakes out of a total of 60 '3's it was tested on, the IWN 4, and the MFN none. These differences of 6 and 10 between the performances of IWN and MFN with that of the CWN are the top two differences over all numerals. The next highest difference is 4 for the numeral '5', and the average difference[21] over all 10 numerals is 1.5 and 2.3, respectively. Why is the CWN making very much higher than average mistakes on '3' compared with the IWN and MFN? It may have to do with the amount of noise in the training data,

[21] This average difference is the average of the sum of absolute difference between the mistakes made by the CWN & IWN, and CWN & MFN, respectively.

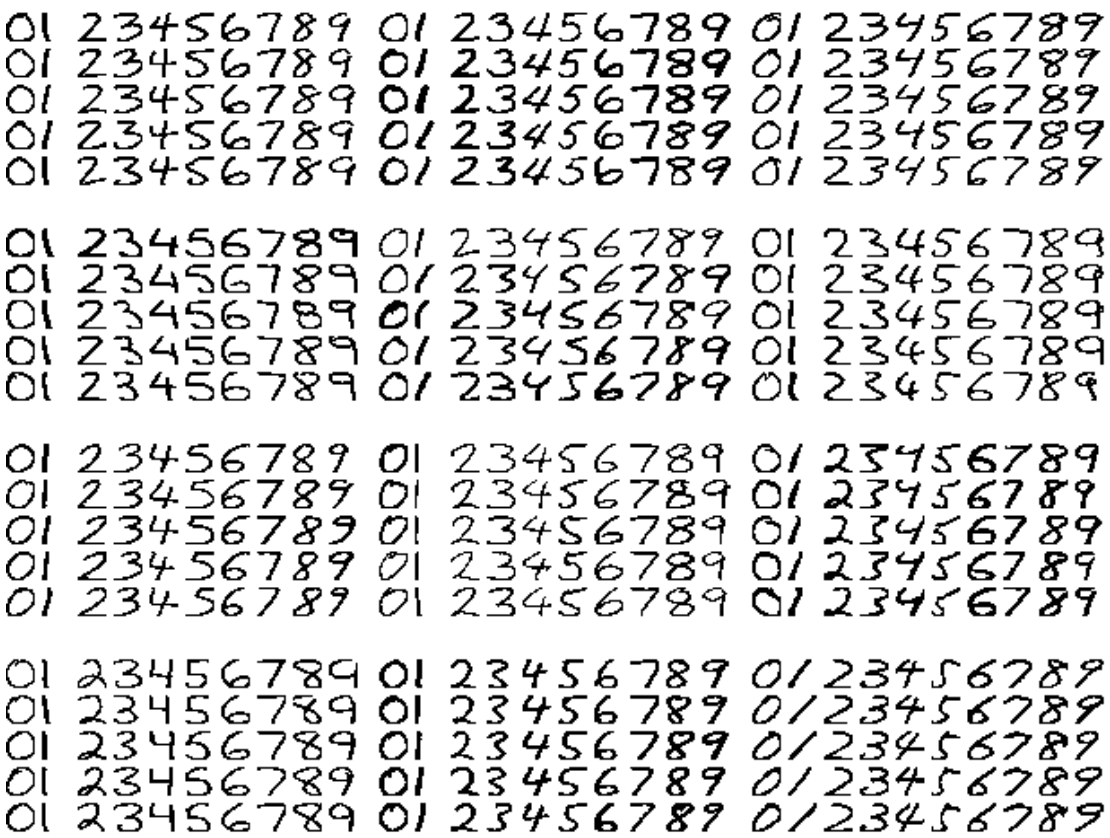


Figure 6.5 Training data for handwritten numeral recognition

and the previously stated hypothesis that networks with discrete synapses outperform the CWN on test data when low-level noise is present in the training data. Visual inspection of the '3' features of Figure 6.7, however, does not prove or disprove this explanation.

The number of zero weights in the best performing IWN and MFN is quite large, 38% and 32%, respectively. There are only 271 non-zero weights in the case of the IWN. As each of these weights is 3 bits deep, the complexity of the network is 813 bits.[22] This means that the network requires approximately 82 bits to store the non-parametric characteristics of each numeral to give a generalisation performance of about 92%. In the case of the MFN, 307 single bit synapses and 21 high-resolution offsets

[22]The effective complexity, in fact, is somewhat lower than this, as only seven of the eight available states with 3 bits are being used.

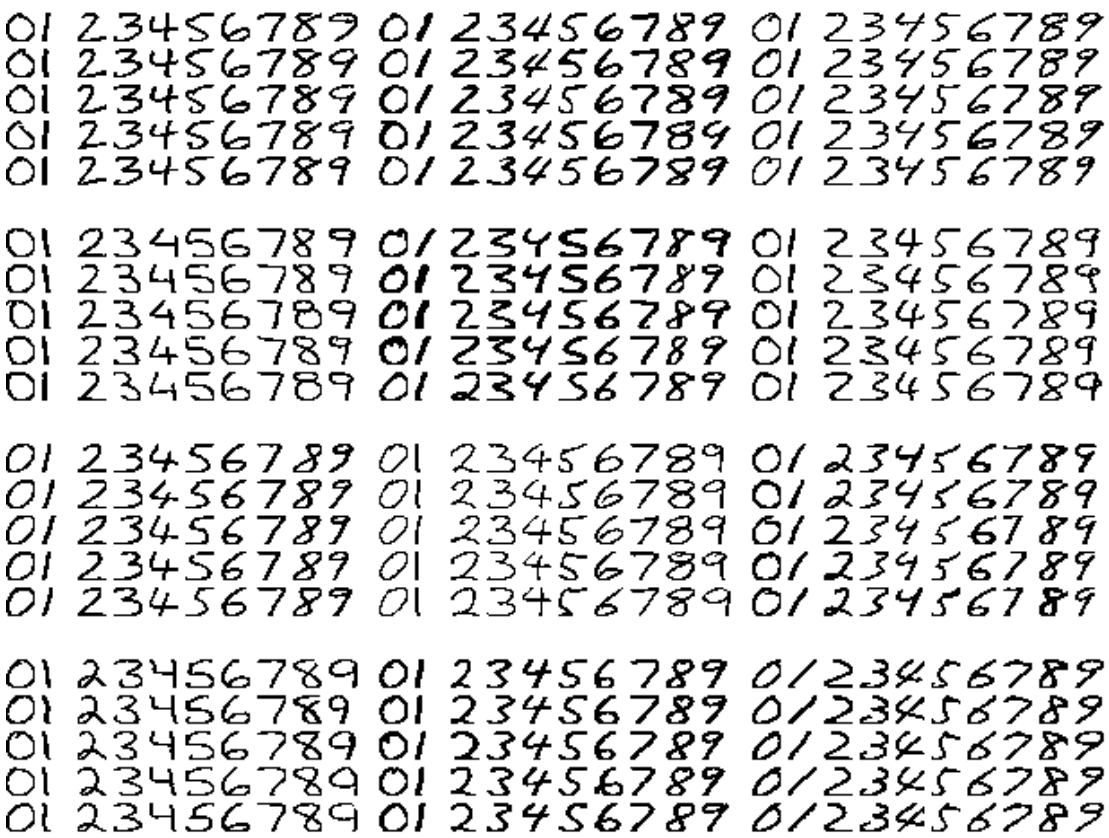


Figure 6.6 Test data for handwritten numeral recognition

are needed. This leads to a figure of 64 or 81 bits per numeral depending upon the fixed-point offset resolution of 16 or 24 bits. The equivalent figures for the CWN are 498 or 746 bits per numeral. The above calculations lead to the inference that for similar generalisation performances, the ranking in terms of storage efficiency is MFN, IWN, and CWN, with MFN being the most efficient.

6.6 Discussion

Experiments were performed to compare the generalisation performances of the three feedforward network paradigms, CWN, IWN, and MFN, in similar circumstances. These experiments used three very different learning benchmarks for the comparison: the MONK's benchmark, an artificial set designed to compare the capabilities of learning algorithms, the 'onset of diabetes mellitus' prediction data set, a realistic set with

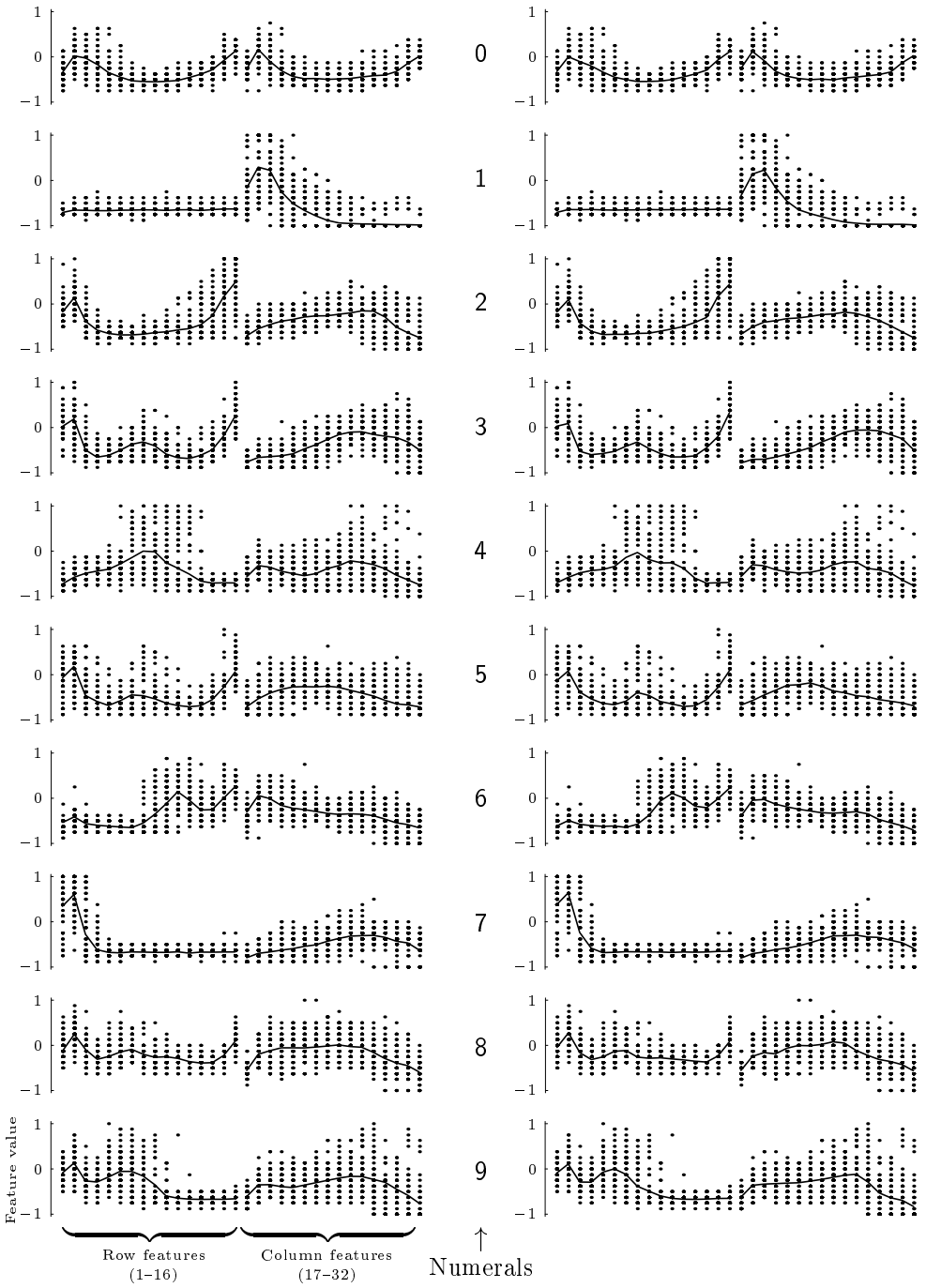


Figure 6.7 Distributions of the individual row and column features obtained after preprocessing the handwritten numeral training (left) and testing (right) data according to the scheme depicted in Figure 6.4. The line passes through the average value of each feature.

Table 6.5 Comparison of generalisation performance on handwritten numeral recognition

Network	Configuration	Zero Weights	Effective Weights	Gen. Perform.
CWN	32:7:10	–	311	92.3%
IWN	32:10:10	169	271	91.8%
MFN	32:11:10	155	328	93.2%

very noisy attributes, and finally the handwritten numeral recognition database, a realistic but very structured data set.

The learning procedure used for these generalisation experiments incorporates an implicit weight elimination mechanism. This feature simplifies the choice of the start-up network configuration for training as the procedure eliminates any superfluous weights, and consequently any surplus hidden neurons, leading to the optimal network of Section 6.4 automatically. Not only that, the number of training epochs was generally smaller for larger start-up configurations.

The IWN was shown not to have the universal approximation capability in Chapter 3, whereas it was conjectured in Chapter 5 that the MIMO MFN is a universal approximator. This suggests that the learning ability, and therefore the generalisation performance, of the former should not be any better than that of the latter. The experimental results of on all three benchmarks support this suggestion. This criterion also suggests that the generalisation performance of MFN and CWN should be similar. The MFN, however, has synapses which are small in magnitude and some of them have zero values. These two factors discourage over-fitting and lead to the learning of smoother and simpler mappings, and hence to better generalisation performance. The experiments were inconclusive in confirming this trend – on the MONK's artificial benchmark the performance was equal, on the very noisy forecasting the onset of diabetes the CWN was better by 0.4%, and on the larger but less noisy handwritten numeral recognition MFN was better by 0.9%. These differences, and those with respect to the IWN, are nevertheless too small to manifest the supremacy of any one of the three paradigms over the other two.

Table 6.6 Classification chart for the test data showing the actual handwritten numerals and the numerals predicted by the three feedforward networks.

	Actual	Predicted Numeral										Errors for	Errors for
Network	Numeral	0	1	2	3	4	5	6	7	8	9	the Numeral	all Numerals
	0	60	0	0	0	0	0	0	0	0	0	0	
	1	0	60	0	0	0	0	0	0	0	0	0	
	2	2	0	53	0	0	3	1	1	0	0	7	
	3	0	0	2	50	0	2	1	0	5	0	10	
CWN	4	0	1	0	0	58	0	0	0	0	1	2	46
	5	3	1	4	6	0	43	0	0	3	0	17	
	6	0	0	1	0	0	0	59	0	0	0	1	
	7	0	0	0	1	0	0	0	59	0	0	1	
	8	0	0	1	3	0	2	0	0	54	0	5	
	9	0	0	0	0	2	0	0	1	0	57	3	
	0	60	0	0	0	0	0	0	0	0	0	0	
	1	0	59	0	0	1	0	0	0	0	0	1	
	2	1	0	52	1	0	3	3	0	0	0	8	
	3	0	0	1	57	0	0	0	1	1	0	4	
IWN	4	0	0	0	0	57	0	0	1	0	2	3	49
	5	0	5	5	6	0	42	0	1	1	0	18	
	6	0	0	0	0	0	0	59	0	1	0	1	
	7	0	0	0	1	0	1	0	58	0	0	2	
	8	0	0	2	3	0	2	0	0	53	0	7	
	9	0	0	0	0	4	0	0	1	0	55	5	
	0	59	0	0	0	0	1	0	0	0	0	1	
	1	0	59	0	0	1	0	0	0	0	0	1	
	2	0	0	54	1	0	2	3	0	0	0	6	
	3	0	0	0	60	0	0	0	0	0	0	0	
MFN	4	0	0	0	0	59	0	0	0	0	1	1	41
	5	5	1	4	10	0	39	0	0	1	0	21	
	6	0	0	0	0	0	0	60	0	0	0	0	
	7	0	0	0	1	0	3	0	56	0	0	4	
	8	0	0	0	2	0	0	2	0	56	0	4	
	9	0	0	0	0	3	0	0	0	0	57	3	

The IWN solution for MONK's problem 3 points towards a potentially very useful feature of discrete-weight networks: the IWN only captures the main features of the training data in its discrete-valued weights. The small number of noisy examples present in the training data cannot perturb the discrete weights, because these weights require relatively large number of variations in the input data in order to jump from one discrete value to another. This noise rejection feature was not conclusively observed for the MFN, probably because not all of its weights are discrete.

In conclusion, the results of this chapter indicate that the IWN and MFN, despite having strong constraints on their weights, have generalisation performances similar to that of the CWN. Moreover, for similar generalisation performances, the ranking in terms of weight storage efficiency in descending order is: MFN, IWN, and CWN. In an application where the number of learning epochs is not of consequence, and the cost and speed of the hardware implementation is the critical factor, both IWN and MFN hold a clear advantage over the CWN due to the storage efficiency of their weights and the elegance of the way they implement their internal multiplication operation.

7

Conclusions and Further Work

7.1 Achievements and Conclusions

Three new ideas were presented in this thesis: feedforward networks having integer-valued weights only, multiplier-free feedforward networks, and the integer-weight learning procedure. Their merits and limitations will now be summarised.

The integer-weight feedforward networks proposed in this thesis take all their weight values from the set $\{-3, -2, -1, 0, 1, 2, 3\}$. These integer weights can be represented by just 3 binary bits. This property reduces the amount of memory required for weight storage in digital-electronic implementations. It also simplifies the digital multiplication operation as multiplying any number with this 3-bit weight requires a maximum of three basic instructions – one shift, one add, and one sign-change. In analogue electronic implementations, where weights can be stored as resistances, the integer-weight scheme requires only 3 distinct resistance values, which streamlines the manufacture by avoiding the expensive trimming schemes required to achieve the precise resistances for conventional networks. In optically implemented networks, where weight values can be represented as grey-scale masks or voltage levels for spatial light modulator, the complexity of implementation is again reduced because of integer valued weights. The

presence of zero valued weights simplifies the design of optical as well as all other forms of hardware.

These networks may have beneficial generalisation properties due to their simpler structure, but the hardware implementation advantage on its own makes them sufficiently attractive to make it worthwhile to investigate their properties. This thesis has provided experimental evidence that these networks are a viable alternative to networks having continuous weights. Although lacking the universal approximation capability, they can implement learning tasks, especially classification tasks, to reasonable accuracies. Experiments indicate that they take relatively longer to train, but have similar generalisation performance to their continuous-weight counterparts.

Discretisation schemes other than integers were also explored. Decision and error surfaces were used to explore the discrete-weight approximation of continuous-weight networks. The results suggest that provided a suitable discretisation interval is chosen, a discrete-weight network can be found which performs as well as a continuous-weight networks, but that it may require more hidden neurons than its continuous-weight counterpart.

The continuous-weight network can store an infinite variety of information, whereas the discrete-weight network's storage is finite and depends upon the resolution of weights. Changing the discretisation scheme of the weights of a fixed-size network, or keeping a fixed discretisation scheme while changing the size of the network, are alternative techniques for determining the amount of network complexity required for approximating the response of a continuous-weight network to a specified tolerance. This extra level of control available on the complexity of discrete-weight networks can be beneficially exploited. Whereas the continuous-weight network has only one parameter, the number of hidden neurons, that can be used to adjust its complexity, the discrete-weight network has two: number of hidden neurons and the weight depth. Using the additional selectivity of the weight depth, the network designer can choose a network with a complexity that matches more closely the complexity of the learning task.

The new multiplier-free feedforward network paradigm proposed in this thesis lends itself to efficient hardware implementation because it does not require a conventional

multiplier for its operation – the expensive and/or slow multiplication operation has been simplified to a single sign-adjustment instruction. This was accomplished by restricting all synaptic values to $\{-1, 0, 1\}$. The number of multiplication operations required for a forward pass in a feedforward network is equal to the number of synapses in that network. Replacement of all these operations with simple sign adjustments can greatly speed up the that operation: in software and in hardware, on sequential systems and on parallel systems. The $\{-1, 0, 1\}$ synapse scheme has the added benefit of storage efficiency. These multiplier-free networks are, in general, expected to be larger than their conventional counterparts in terms of the number of hidden neurons, but should be more compact in hardware.

It is possible that the severity of the $\{-1, 0, 1\}$ synapse restrictions may weaken the approximation capability of this network, but experiments on classification tasks indicate otherwise. Comfort is also provided by the theoretical result on approximation in $C(\mathbb{R})$ presented in this thesis. That result guarantees that networks with input-layer synapses from the set $\{-1, 1\}$, no output-layer synapses, and continuous offsets can approximate all continuous function of one variable to any desired accuracy.

The new integer-weight learning procedure proposed in this thesis starts off like conventional error backpropagation, but becomes more and more discretised in its behaviour as the network gets closer to an error minimum. It does not, however, make the assumption that the acceptable error minima for both the integer-weight network and the continuous-weight one are located at the same location. Mainly based on steepest descent, it also has a perturbation mechanism to avoid getting trapped in local minima. Moreover, it has a separate mechanism for rounding off 'near integers'. This same integer-weight learning procedure was used to train the synapses of the multiplier-free networks – the offsets were trained using conventional error backpropagation. The structure of this procedure is such that it can be readily adapted for weight discretisation schemes other than $\{-3, -2, -1, 0, 1, 2, 3\}$ and $\{-1, 0, 1\}$.

The learning procedure incorporates an implicit weight elimination mechanism. This feature simplifies the choice of the start-up network configuration for training as the procedure eliminates any superfluous weights, and consequently any surplus hid-

den neurons, leading to a network of an optimal configuration automatically. Moreover, the number of training epochs was generally smaller for larger-than-necessary start-up configurations.

On the other hand, the new learning procedure requires high-resolution arithmetic which makes it unsuitable for in-situ learning. Many commercial feedforward network applications, however, do not require in-situ training. The possibly large number of learning epochs required for integer-weight networks, compared with continuous-weight ones, should not be a deterrent to their use as, in many practical applications, the learning period occupies only a small fraction of a network's lifetime usage. When products incorporating such applications are mass produced, the length of training time becomes an insignificant part of the combined design and manufacture process.

Experiments were performed to compare the generalisation performances of the three feedforward network paradigms, conventional, integer-weight, and multiplier-free, in similar circumstances. These experiments used three very different learning benchmarks for the comparison: the MONK's benchmark, an artificial set designed to compare the capabilities of learning algorithms, the 'onset of diabetes mellitus' prediction data set, a realistic set with very noisy attributes, and finally the handwritten numeral recognition database, a realistic but very structured data set.

The results indicate that the integer-weight and multiplier-free networks, despite having strong constraints on their weights, have generalisation performances similar to that of their conventional counterpart. In an application where the number of learning epochs is not of consequence, and the size and speed of the hardware implementation is the critical factor, these networks with constrained weights hold a clear advantage over the conventional network due to the storage efficiency of their weights and the elegance of the way they implement their internal multiplication operation.

7.2 Future Work

The following five projects are suggested to further the line of investigation presented in this thesis.

More work is needed on discrete but non-integer weights, and the investigation into the improvement in approximation capabilities with the reduction of the discretisation interval. A study analysing the tradeoff between the increase in the cost of hardware due to the decrease in the discretisation interval and the resultant improvement in approximation capability will be very useful.

The multiplier-free network was shown to be a universal approximator on the space of continuous function of one variable in Chapter 5. The extension of this result to the space of continuous functions of many variables will be very valuable.

There is an inverse relationship between the number of hidden neurons required to learn a particular task with an acceptable error and the weight depth. A procedure that could simultaneously vary these two parameters during training will be beneficial in determining the most cost-effective combination for implementation in hardware.

The Bayesian framework has been employed to automatically determine the optimal values for weight decay parameters during training to achieve optimal generalisation performance [141]. As weight discretisation is similar to weight decay, in that it is also a constraint on weights [140], the optimal values for weight discretisation parameters can also be determined using Bayesian techniques which will result in optimal generalisation performance for these constrained-weight networks.

The fast forward-pass capability of a multiplier-free network, implemented on an inexpensive microcontroller, can be exploited for the non-linear closed-loop control of a magnetic-bearing spindle system which has an unstable open-loop response. This spindle can replace the conventional ball-bearing spindle in a high speed machining process to achieve better cutting stability [158].

Glossary*

Activation function is the transform applied to the weighted sum of inputs plus offset for computing the output of a neuron. Also known as the squashing function.

Affine group invariance. The property of a group due to which it stays unchanged after the application of an affine transform.

Affine transform is a transform from the set of rotations, shifts, scalings, or any combinations thereof.

Algebra. A set of functions $\mathcal{A}$ is an algebra if $f, g \in \mathcal{A},\ \vartheta \in \mathbb{R} \Longrightarrow f + g \in \mathcal{A},\ f \cdot g \in \mathcal{A}$, and $\vartheta f \in \mathcal{A}$.

Alopex is a stochastic learning procedure which uses local correlations between changes in individual weights and changes in the cost function to update weights. This procedure predates the current resurgence in neural network learning by more than a decade and was originally proposed for mapping visual receptive fields [146].

Approximation property, Universal, is the ability of a set functions to approximate a specific class of functions to any desired accuracy.

Approximation property, Best, is the property of an approximation scheme on a set of functions to select a function that is at a minimum 'distance' from the function to be approximated.

Some of the terms are not explicitly mentioned in this dissertation but are included because of their usefulness in understanding the cited literature.

ART-EMAP is ARTMAP with added spatial and temporal evidence accumulation processes [25].

ARTMAP is a supervised learning procedure explicitly based on neurobiology.

ARTMAP-IC is ARTMAP with an instance counting procedure and a match tracking algorithm [25].

Attribute is an element of the input vector. Also known as a feature.

Autoassociator A system for which the desired output response is the same as the input.

Backpropagation, Error is an procedure in which the difference between the actual and desired responses of the neurons in the output layer is minimised using the steepest-descent heuristic.

Balanced data set is set in which all classes are equally represented.

Bayesian classifier assigns a class to an object in such a way that the expectation value of misclassification is minimised. Also known as the minimum risk classifier, and belief network.

Bayesian statistics differs from the conventional 'frequentist' approach to statistics in that it allows the probability of an event to be expressed as 'degree of belief' in a particular outcome instead of basing it solely on a set of observations [15].

Bayes's theorem allows prior estimates of the probability of an event to be revised in accordance with new observations. It states that probability of an event A given another event B, P(A|B), is equal to P(B|A)P(A)/P(B).

Black-hole mechanism is a rounding mechanism for 'nearly discrete' weights.

Black-hole radius. If the value of a weight gets within this radius of a discrete value, it becomes that discrete value.

Bootstrap. A random sample is selected by sampling with replacement from the data set and is used to train the network. The trained network is then tested on the

remaining data. This procedure is repeated a large number of times. The average of all such test errors is an estimate of the generalisation performance metric.

Borel measurable functions. Just about all functions that one may encounter are Borel measurable. Functions that are not Borel measurable do exist but are known to mathematicians only as mathematical peculiarities.

Cascade-correlation learning method starts with a network without any hidden neurons and systematically increases their number during training until the required performance is achieved.

Cauchy sequence is a sequence $\{a_n\}$ of real numbers for which for every $\epsilon > 0$, there exists a positive integer n_0 such that $|a_m - a_n| < \epsilon$, whenever $m > n \geq n_0$.

Classification is a task in which the desired responses are restricted to a finite set of values.

Closure. $Cl(A) = A \cup \{\text{Limit points of } A\}$.

Compact set. A closed and bounded subset of $\mathbb{R}^d$. Also known as a compact.

Compact in $\mathbb{R}^n$. A is a compact in $\mathbb{R}^n$ if it is a subset of $\mathbb{R}^n$, is a closed set, and is a bounded set [90].

Convergence, Pointwise. If $\{a_n\}$ is a sequence of non-random real variables then a_n converges to a, i.e., $a_n \to a$ as $n \to \infty$, if there exists a real number a such that for any $\epsilon > 0$, there exists an integer N_ϵ sufficiently large that $|a_n - a| < \epsilon$ for all $n \geq N_\epsilon$ [155]. Also known as deterministic convergence.

Convergence in distribution. If $\{\hat{a}_n\}$ is a sequence of random variables having a distribution function $\{F : F_n(a) \equiv P[\hat{a}_n \leq a\}$ then $\hat{a}_n$ converges to F in distribution, i.e. $\hat{a}_n \xrightarrow{d} F$, iff $|F_n(a) - F(a)| \longrightarrow 0$ for every continuity point a of F [155].

Convergence in probability. If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges to a in probability, i.e. $\hat{a}_n \xrightarrow{P} a$, if there exists a real number a such that for any $\epsilon > 0$, $P[|\hat{a}_n - a| < \epsilon] \to 1$ as $n \to \infty$. Also known as weak convergence [155].

Convergence in the mean. If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges to a in the mean if $\lim_{a_n \to \infty} E\{|\hat{a}_n - a|\} = 0$, where $E\{a\}$ represents the estimated value of a.

Convergence in the mean squared sense. If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges to a in the mean squared sense if $\lim_{a_n \to \infty} E\left\{|\hat{a}_n - a|^2\right\} = 0$, where $E\{a\}$ represents the estimated value of a.

Convergence with probability 1. If $\{\hat{a}_n\}$ is a sequence of random variables then $\hat{a}_n$ converges with probability 1 to a, i.e. $\hat{a}_n \to a$ as $n \to \infty$ or $\hat{a}_n \stackrel{P=1}{\longrightarrow} a$, if $\lim_{a_n \to \infty} P[\hat{a}_n = a] = 1$ for some real number a. Also known as almost sure convergence, convergence almost everywhere, and strong convergence. [155]

Convergence, Uniform The property that all of a family of functions or series on a given set converge at the same rate throughout the set; that is, for every $\varepsilon > 0$ there is a single N such that for all points in the set , $|f_m(x) - f_n(x)| < \varepsilon$ for all $m, n > N$ and similarly for uniform convergence as x tends to a value a [20].

Cost function is the quantity that is to be minimised in an optimisation experiment. In the case of feedforward networks this quantity is usually the RMS error in the output of the network. Also known as error measure.

Cover of a set A is a collection of sets $\{T_i\}$ whose union contains A [90].

Cover, Open. It is an open cover of $\{T_i\}$ if each T_i is open [90].

Cover, Sub- of a given cover is a subcollection whose union also contains A [90].

Cross-validation, n-fold. The data set is divided equally into k randomly selected, mutually exclusive subsets called folds. $k-1$ networks are trained sequentially on all combinations of $k-1$ folds, while the performance of the trained networks is tested on the one remaining folds. The average of $k-1$ such errors is an estimate of the generalisation performance metric.

Decision sensitivity is the likelihood that an event will be detected if it occurs. It is the ratio of true positives to the sum of true positives and false negatives. This

metric is especially of importance when it is critical that a an event be detected. Also known as True Positive Ratio [34].

Decision specificity is the likelihood that the absence of an even is detected given that it is present. It is the ratio of the true negatives to all negatives. Also known as True Negative Ratio [34].

Decision surface is the plot of the response of an output neuron with respect to the inputs.

Denseness. A set A is dense in a set S if $A \subset S$ and $Cl(A) = S$.

Denseness, Uniform. A set of functions $\mathcal{A}$ is uniformly dense in $C(\mathbb{R}^d)$ on the compact set $K \subset \mathbb{R}^d$ if for all $f \in C(\mathbb{R}^d)$ and every $\varepsilon > 0$ there exists $\hat{f} \in \mathcal{A}$ such that $\sup\{|f(\mathbf{x}) - \hat{f}(\mathbf{x})| : \mathbf{x} \in K\} < \varepsilon$ [137].

Denseness, Uniform on compacta. A set of functions $\mathcal{A}$ is uniformly dense on compacta in $C(\mathbb{R}^d)$ if it is uniformly dense on every compact subset of $\mathbb{R}^d$ [137].

Disjunctive normal form (DNF). The form of a logical expression consisting of a single conjunction $(\cdot)$ of a set of disjunctions$(+)$. All logical expressions are expressible in this form [61].

Effective sample size (for classification learning tasks) is the number of examples representing the smallest classification group [95].

EM algorithm. Expectation Maximisation algorithm calculates the probability density of observations based on parameters and not observations [117].

Epoch is the cycle in which all examples in the training set are presented to the network.

Ergodic process. A random process is ergodic if its ensemble and temporal averages are the same.

Error surface is the plot of the cost-function with respect to all of the weights in a network.

Feedforward network consists of a layer of inputs, zero or more layers of hidden neurons, and an output layer of neurons. Generally all neurons in adjacent layers are fully connected to each other with feedforward synapses only. There are no intra-layer synapses. Also known as the multilayer perceptron.

Fit, Over- An over-fit is due to the trained network having a higher complexity than the concept embedded in the training data. Also known as memorisation and over-specialisation.

Fit, Under- An under-fit is caused by the trained network having a complexity lower than that of the concept embedded in the training data.

Forward pass. The process by which a network computes the output vector in response to an input vector. Also known as recall.

Function, Analytic. $f \in C(\mathbb{R})$ is analytic at $a \in \mathbb{R}$ with a radius of convergence $r > 0$ if there is an infinite sequence of real numbers, $\{c_n\},\ n \geq 0$, such that for $|x-a| < r$, $\sum_{n=0}^{\infty} c_n(x-a)^n$ converges and $f(x) = \sum_{n=0}^{\infty} c_n(x-a)^n$ [137]

Function, Superanalytic. $f \in C(\mathbb{R})$ is superanalytic at $a \in \mathbb{R}$ with a radius of convergence $r > 0$ if there is an infinite sequence of real numbers, $\{c_n\},\ n \geq 0$, and if for every $n \geq 1,\ c_n \neq 0$, such that for $|x-a| < r,\ \sum_{n=0}^{\infty} c_n(x-a)^n$ converges and $f(x) = \sum_{n=0}^{\infty} c_n(x-a)^n$ [137]

Function approximation is a task in which the desired output values are continuous. Also known as regression.

Functional is a scalar-valued continuous linear function defined on a normed linear space.

Functional, Linear on a linear space E over $\mathbb{R}$ is a linear transformation of E into $\mathbb{R}$.

Functional, linear, Bounded. A bounded linear transformation of a normed linear space E over $\mathbb{R}$ into the normed linear space $\mathbb{R}$ is called a bounded linear functional on E.

Generalisation performance is the accuracy of decision of a trained network on a set of data which is similar to but not the same as the training data set.

Hahn-Banach theorem. Let M be a linear subspace of a normed linear space N, and let f be a functional defined on M. Then f can be extended to a functional f_0 defined on the whole space N such that $\|f_0\| = \|f\|$.

If M_c is a closed linear subspace of N and x_0 is a vector not in M_c, then there exists a functional f_0 in the conjugate space N^* such that $f_0(M_c) = 0$ and $f_0(x_0) \neq 0$ [133].

Hebbian learning. The main idea behind Hebbian learning is that the synapse between two neurons should be strengthened if they fire simultaneously.

Hidden layer is the layer of neurons which is not directly connected to the network inputs or outputs.

Homogeneity property. A set of functions $\mathcal{A}$ fulfils the homogeneity property if $f \in \mathcal{A}$ and $\vartheta \in \mathbb{R} \Longrightarrow \vartheta f \in \mathcal{A}$.

Inequality, triangle. $|a + b| \leq |a| + |b|$

***k*-nearest neighbours** is a clustering algorithm that minimises the the sum of squares of distances between the training data and k points.

***L*$_p$-norm** measures is a popular form of the cost function for feedforward networks.

$$E_o(\mathbf{W}) = \left(\sum_{j=1}^{J} |t_j - o_j|^p \right)^{\frac{1}{p}}, \qquad p = 1, 2, \ldots \infty, \tag{7.1}$$

where t_j is the target or desired value of the jth output, and o_j is its value computed by the network.

Learning is the process in which a feedforward network is forced to adjust its weights such that the network's response to a given input vector becomes closer to the desired response.

Learning, Batch. The type of learning during which weights are updated at the end of every epoch. Also known as off-line learning.

Learning, In-situ differs from on-line learning in that the former is the property of a *network* requiring the deployed network to have adaptive weights, whereas the later is a property of the *learning procedure*, requiring the weights to be updated on the presentation of every example.

Learning, On-line. The type of learning during which weights are updated after the presentation of every training example. Also known as pattern and incremental learning.

Learning, Supervised The learning process in which a system's internal parameters are modified in order to minimise the error in its output with respect to a desired value.

Learning, Unsupervised The learning process in which a system's internal parameters are modified so that similar input patterns result in similar outputs.

Learning rate determines the size of the weight modification at each training step.

Likelihood is the probability density of observations calculated from parameters and not observations [117].

Limit Point. A point p is a limit point of A if every neighbourhood of p contains a point $q \neq p$ such that $q \in A$.

Linear separability. The property of a classification task by which the members of one class can be separated from the ones from all other classes by a single hyperplane.

Loading problem, The. The problem of finding the optimal weight values for a given network such that the network performs the required mapping.

Logistic discriminant analysis chooses classification hyperplanes with respect to maximising a conditional likelihood cost-function and not optimising a quadratic cost-function which is the case for linear discriminant analysis [93].

Margin, ε_n. Error in the output of a neuron is not backpropagated if it is within this small margin [139].

Minima, Global. The points of minimum error on an error surface.

Minima, Local. The points of zero gradient on an error surface which are not global minima.

Mixture representation of data use a linear combination of Gaussian distributions to represent arbitrary distributions [10].

Momentum is a training parameter used in a very common variation on standard error backpropagation learning procedure. It controls the effect of the last weight modification on the current weight update.

n-layer network is a feedforward network with $n-1$ hidden layers.

NP-complete problems. (non-polynomial time problems) The time required to find the optimal solution for this class of problems grows exponentially with the size of the problem. Also known as intractable problems.

Neural network, Artificial is a set of interconnected artificial neurons.

Neuron, Artificial is the fundamental processing element in an artificial neural network. It performs a weighted sum of its inputs, adds the offset value to that sum, and then outputs a certain transform of that sum. Also known as node and processing element (PE).

Ockham's Razor is the conjecture that if, for a given problem, two solutions with similar performances are available then the one with the lower computational complexity should be preferred.

Offset is the value added to the weighted sum before the transform is applied to compute a neuron's output. Also known as threshold and bias.

Over-trained networks have a complexity higher than what is required to learn the concept embedded in training data. They act as look-up tables for the training

data and are poor generalisers.

Perceptron is a feedforward network with no hidden neurons.

Probability, Prior is the probability assigned to an event in advance of any empirical evidence. Also known as 'a priori' probability.

Probability, Posterior is the probability assigned to an event based on observations. Also known as 'a posteriori' probability.

Projection pursuit regression is a generalisation of the feedforward network in that it allows more than one type of activation function in the hidden layer. These non-homogeneous activation functions are data-dependent and constructed during learning [66].

Regularisation A class of methods designed to avoid overfitting to the training data by enforcing smoothness of the fit.

Ridge regression. The precision of least-squares estimates gets worse with with an increase in dependence between the input variables. Ridge regression estimators are more precise in those situations and are obtained as the estimators whose distance to an ellipsoid (the 'ridge') centred at a least-squares estimate from the origin of a parameter space is a minimum [46].

Ridging, Constrained. Optimisation procedure in which some norm of the weights is constrained to a specific value [130].

Ridging, Penalised. Optimisation procedure in which the cost function is augmented by a penalty term [130].

Ridging, Smoothed. Optimisation procedure in which noise is introduced in the inputs [130].

Riesz representation theorem. Let x^* be a bounded linear functional on the Banach space $C_{\mathbb{R}}([a,b])$. Then there is a real-valued function α of bounded variation on $[a,b]$ such that $x^*(f) = \int_a^b f\, d\alpha$ for all $f \in C_{\mathbb{R}}([a,b])$. Further, if x^* is a positive linear functional, then α is increasing on $[a,b]$ [23].

RMS error, $E_{o_{rms}}$, is computed by summing the output layer errors for all examples in a training or test set, dividing the sum by the total number of examples and the number of the output layer neurons, and taking the square root of the resultant. The output layer error is computed by summing the squares of the individual neuron errors with respect to the desired output. An individual output-layer neuron's error is set to zero if it is less than the margin.

Sampling with replacement may result in successive samples being not mutually exclusive, some of the examples may never appear in any of the samples, and there may be repetitions within an individual sample.

Separates points. A family of functions $\mathcal{A}$ separates points on a set S if for every $x, y \in S, x \neq y$, there exists $f \in \mathcal{A}$ such that $f(x) \neq f(y)$ [136].

Set, Closed. A subset M of metric space N is a closed set if it contains each of its limit points.

Set, Finite. A is finite if all of its elements can be displayed as $\{a_1, a_2, ..., a_n\}$ for some integer n [90].

Set, Open is the subset G of the metric space X if each point of G is the centre of some open sphere contained in G.

Shattered. If a set of functions F includes all possible dichotomies on a set S of points, then S is said to be shattered by F. [6]

Shrinkage. The difference between the training set accuracy of a network and its accuracy on a test set.

Sigmoidal functions. Definitions vary but are generally taken to be bounded, monotone, and continuous, e.g. logistic and $\tanh(\cdot)$ functions.

Simulated annealing is a stochastic optimisation technique inspired by the physical process of annealing.

Skip-layer synapses. Synapses connecting neurons in two non-adjacent layers. Also known as short-cut synapses [78]. Known as main effects in the statistical literature [127].

Smoothing spline modelling is piecewise approximation by polynomials of degree n with the requirement that the derivatives of the polynomials are continuous up to degree $n-1$ at the junctions [20].

Softmax. The purpose of the softmax activation function is to make the sum of the output neuron responses equal to one, so that the outputs are interpretable as posterior probabilities. Also known as the multiple-logistic function.

Space, Banach is a complete normed linear space.

Space, Compact is a topological space in which every open cover has a finite subcover.

Space, Complete metric is a metric space in which every Cauchy sequence is convergent.

Space, Conjugate. N^* is the set of all continuous linear transforms of the normed linear space N into $\mathbb{R}$.

Space, Eucledian is the metric space $(\mathbb{R}, d)$ such that $d(x,y) = \left(\sum_{r=1}^{n} (x_r - y_r)^2\right)^{1/2}$.

Space, Hausdorff is a topological space, T, in which any two given distinct points $x,\ y$ are such that there exist disjoint open subsets $U,\ V$ containing $x,\ y$ respectively.

Space, L_p consists of all measurable functions f defined on a measure space M with measure m which are such that $|f(x)|^p$ is integrable, with the norm taken as $\|f\|_p = \left(\int |f(x)|^p dm(x)\right)^{1/p}$.

Space, Normed linear over $\mathbb{R}$ is a pair $\{E, \|\cdot\|\}$, where E is linear space over $\mathbb{R}$ and $\|\cdot\|$ is a norm on E. Normed linear space is a metric space with the metric being $\|x-y\|$.

Space, Topological is a pair (X, T), where X a non-empty set and T is collection of subsets of X such that the subsets are closed under union and intersection

operations.

Span of a set of functions. For any function $f \in C(\mathbb{R})$ and $r > 0$, $f|(-r,r)$ denotes the restriction of f to the interval $(-r,r)$, and for any class of function $\mathcal{F}$, $\mathcal{F}|(-r,r)$ denotes $\{f|(-r,r) : f \in \mathcal{F}\}$. For any $\mathcal{F}$ defined on a set $\mathcal{O}$, the span of $\mathcal{F}$, $sp(\mathcal{F})$, denotes the closure of the set of finite linear combinations of elements of $\mathcal{F}$ in the topology of uniform convergence on compact subsets of $\mathcal{O}$ [137].

Sphere, Open. $S_r(x_0)$ with centre x_0 and radius r is the subset of the metric space X with metric D defined by $S_r(x_0) = \{x : d(x, x_0) < r\}$.

Stationary, strongly. If a random variable X is strongly stationary then the distribution of $X(t)$ is independent of the time t [45].

Subset, Proper. A is a proper subset of B if $A \subset B$ and $B \not\subset A$.

Subspace, Linear is the non-empty subset, M, of a linear space if $(x+y) \in M$ whenever $x \in M$ and $y \in M$, and if $\alpha x \in M$ whenever $x \in M$, where α is a scalar.

Supremum is the least upper bound for a set.

Synapse is a measure of the effect that a neuron's output has on the output of another neuron at the other end of the synapse. Also known as connection, edge, and weight.

Testing is the process of verifying the function of a trained network against a set of examples which is different from the training examples set.

Train-and-test. A random sample containing one half of the total number of examples is selected. This subset is used to train the network while the remaining examples are used to test the network once it has been trained. The performance of the trained network on the test set is an estimate of the generalisation performance metric.

Training See *Learning*.

Training example is a pair: an input vector, and the desired response to that input vector.

Vanishes at no point. A family of functions $\mathcal{A}$ vanishes at no point of the set S if for each $x \in S$ there exists $f \in \mathcal{A}$ such that $f(x) \neq 0$ [136].

Weight is the value of a synapse or an offset.

Weight decay is a common regularisation technique used in feedforward network training in which the cost-function is augmented with a term which penalises large weight values.

Weight depth is the number of binary bits in a weight.

Weight elimination is a regularisation technique used in feedforward network training in which the cost-function is augmented with a term which penalises the number of non-zero weights [51, 81, 110, 152].

Weight perturbation is a hardware-friendly alternative to BP learning. In this method, all of the weights are perturbed in turn and the associated change in the output of the network is used to approximate local gradients [59].

Weight sharing is a regularisation technique used in feedforward network training in which the cost-function is augmented with a term which penalises the number of independent weights [105, 122].

References

[1] 1040EZ: Income tax return for single and joint filers with no dependents. Internal Revenue Service, Department of Treasury, United States of America, 1994. Available as `ftp://ftp.fedworld.gov/pub/irs-pdf/f1040ez.pdf`.

[2] Anderson, M. Private communication, July 1996.

[3] Babri, H. A. and Tong, Y. Deep feedforward nets: Applications to pattern recognition. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1422–1426, Washington, DC, June 1996. IEEE Press, New York, NY.

[4] Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, **IT-39**:930–945, 1993.

[5] Battiti, R. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, **4**:141–166, 1992.

[6] Baum, E. B. On the capabilities of multilayer perceptrons. *Journal of Complexity*, **4**:193–215, 1988.

[7] Baum, E. B. and Haussler, D. What size net gives valid generalization? *Neural Computation*, **1**:151–160, 1989.

[8] Bennett, P. H., Burch, T. A., and Miller, M. Diabetes mellitus in American (Pima) Indians. *Lancet*, **2**:125–128, 1971.

[9] Bioch, J. C., van der Meer, O., and Potharst, R. Classification using Bayesian neural nets. In *Proceedings of the IEEE International Conference on Neural*

Networks, volume 3, pages 1488–1493, Washington, DC, June 1996. IEEE Press, New York, NY.

[10] Bishop, C. M. Mixture density networks. Technical report NCRG/4288, Department of Computer Science and Applied Mathematics, Aston University, Birmingham, England, February 1994.

[11] Bishop, C. M. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[12] Bishop, C. M. *Neural Networks for Pattern Recognition*, chapter 3. In [11], 1995.

[13] Bishop, C. M. *Neural Networks for Pattern Recognition*, chapter 8. In [11], 1995.

[14] Bishop, C. M. *Neural Networks for Pattern Recognition*, chapter 7. In [11], 1995.

[15] Bishop, C. M. *Neural Networks for Pattern Recognition*, chapter 1. In [11], 1995.

[16] Bishop, C. M. *Neural Networks for Pattern Recognition*, chapter 9. In [11], 1995.

[17] Bishop, C. M., Haynes, P. S., Smith, M. E. U., Todd, T. N., and Trotman, D. L. Real-time control of a tokamak plasma using neural networks. *Neural Computation*, **7**(1):206–217, 1995.

[18] Blum, A. L. and Rivest, R. L. Training a 3-node neural networks is NP-complete. *Neural Networks*, **5**:117–127, 1992.

[19] Borggaard, C., Madsen, N. T., and Thodberg, H. H. In-line image analysis in the slaughter industry, illustrated by beef carcass classification. Reference number 31.616, manuscript number 1336E, Danish Meat Research Institute, Roskilde, Denmark, June 1996.

[20] Borowski, E. J. and Borwein, J. M. *Dictionary of Mathematics*. HarperCollins, Great Britain, 1989.

[21] Boser, B., Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. In *Proceedings of the Computational Learning Theory ACM Workshop*, Pittsburgh, PA, 1992.

[22] Brause, R. W. The error-bound descriptional complexity of approximation networks. *Neural Networks*, **2**(6):177–187, 1993.

[23] Brown, A. L. and Page, A. *Elements of Functional Analysis*. Van Nostrand Reinhold, London, 1970.

[24] Buckley, J. M. and Richardson, M. B. Control of a copper laser using neural networks. *IEE Computing and Control Engineering Journal*, **7**(3):145–152, June 1996.

[25] Carpenter, G. A. and Markuzon, N. ARTMAP-IC and medical diagnosis: Instance counting and inconsistent cases. Technical report CAS/CNS-96-017, Department of Cognitive and Neural Systems, Boston University, Boston, MA, May 1996.

[26] Carroll, S. M. and Dickinson, B. W. Constructing the neural nets using the Radon transform. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 607–611, Washington, DC, 1989. IEEE Press, New York, NY.

[27] Chen, T., Chen, H., and Liu, R. Approximation capability in $C(\bar{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems. *IEEE Transactions on Neural Networks*, **1**(6):25–30, 1995.

[28] Chieueh, T. D. and Goodman, R. M. Learning algorithms for neural networks with ternary weights. In *First Annual Meeting of International Neural Networks Society*, Boston, MA, September 1988. Abstract in *Neural Networks*, **1**:166, 1988.

[29] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, **6**:303–314, 1989.

[30] Denker, J. S. and Wittner, B. S. Network generality, training required, and precision required. In Denker, J. S., editor, *Neural Networks for Computing*, pages 219–222, Snowbird, UT, 1986. American Institute of Physics, New York, NY.

[31] Drucker, H. and Le Cun, Y. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, **3**:991–997, 1992.

[32] Dugundji, J. *Topology*. Allyn and Bacon, Boston, MA, 1966. Theorem XI.10.1.

[33] Dundar, G. and Rose, K. The effect of quantization on multilayer neural nets. *IEEE Transactions on Neural Networks*, **6**(6):1446–1451, 1995.

[34] Eberhart, R. C., Dobbins, R. W., and Hutton, L. V. Performance metrics. In Eberhart, R. C. and Dobbins, R. W., editors, *Neural Network PC Tools: A Practical Guide*, chapter 7, pages 161–176. Academic Press, London, England, 1990.

[35] Fahlman, S. E. Fast-learning variations on back-propagation: An empirical study. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, Pittsburgh, PA, 1989. Morgan Kaufmann, San Mateo, CA.

[36] Fahlman, S. E. The Cascade-correlation learning algorithm on the MONK's problems. In Thurn et al. [143], chapter 10.

[37] Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[38] Fausett, L. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*, chapter 1. Prentice Hall, Englewood Cliffs, NJ, 1994.

[39] Fausett, L. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*, chapter 6. In [38], 1994.

[40] Fiesler, E., Choudry, A., and Caulfield, H. J. A weight discretization paradigm for optical neural networks. In *Proceedings of the International Congress on Optical Science and Engineering*, pages 164–173, Bellingham, Washington, 1990. SPIE.

[41] Funahashi, K. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**:183–192, 1989.

[42] Gallant, S. I. *Neural Network Learning and Expert Systems*, chapter 11. MIT Press, Cambridge, MA, 1993.

[43] Gallant, S. I. *Neural Network Learning and Expert Systems*, chapter 13. In [42], 1993.

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

[45] Grimmett, G. R. and Stirzaker, D. R. *Probability and Random Processes*. Clarendon, Oxford, England, second edition, 1992.

[46] Gruber, M. H. J. *Regression Estimators: A Comparative Study*. Academic Press, London, 1990.

[47] Guyon, I., Poujaud, I., Personnaz, L., Dreyfuss, G., Denker, J., and Le Cun, Y. Comparing different neural network architectures for classifying handwritten digits. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 127–132, Washington, DC, 1989. IEEE Press, New York, NY. The numeral database has been kindly made available by Isabelle at `ftp://-hope.caltech.edu/pub/mackay/data/att.database`.

[48] Guyon, I., Vapnik, V., Boser, B., and Solla, S. Structural risk minimisation for character recognition. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann, San Mateo, CA, 1992.

[49] Hardy, G. H. and Wright, E. M. *An Introduction to the Theory of Numbers*, chapter 23. Oxford University Press, Oxford, England, fifth edition, 1979.

[50] Hardy, G. H. and Wright, E. M. *An Introduction to the Theory of Numbers*, chapter 23. In [49], fifth edition, 1979. Theorem 439.

[51] Hassibi, B., Stork, D. G., and Wolf, G. J. Optimal brain surgeon and general network pruning. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 293–299, San Francisco, CA, 1993.

[52] Haykin, S. *Neural Networks: A Comprehensive Foundation*, chapter 1. Maxwell Macmillan, New York, NY, 1994.

[53] Haykin, S. *Neural Networks: A Comprehensive Foundation*, chapter 6. In [52], 1994.

[54] Haykin, S. *Neural Networks: A Comprehensive Foundation*, chapter 8. In [52], 1994.

[55] Hecht-Nielsen, R. Theory of the backpropagation neural network. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 593–606, Washington, DC, 1989. IEEE Press, New York, NY.

[56] Heemskerk, J. N. H. *Neurocomputers for Brain-Style Processing. Design, Implementation and Application.* PhD thesis, Unit of Experimental and Theoretical Psychology, Leiden University, The Netherlands, 1995. A draft version of Chapter 3 is available as `ftp://ftp.mrc-apu.cam.ac.uk/pub/nn/murre/neurhard.ps`.

[57] Hinton, G. E. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst, MA, 1986. Erlbaum, Hillsdale, NJ. Reproduced in *Parallel Distributed Processing: Implications for Psychology and Neurobiology*, Morris, R. G. M. editor. Oxford University Press, Oxford, England, 1989.

[58] Hoehfeld, M. and Fahlman, S. E. Learning with limited numerical precision using the cascade-correlation algorithm. Preprint, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1991.

[59] Hollis, P. W. and Paulos, J. J. A neural network learning algorithm tailored for VLSI implementation. *IEEE Transactions on Neural Networks*, **5**(5):784–791, 1994.

[60] Holts, J. H. and Hwang, J.-. N. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, **42**(3):281–290, 1993.

[61] Horne, B. G. and Hush, D. R. On the node complexity of neural networks. *Neural Networks*, **7**(9):1413–1426, 1994.

[62] Hornik, K. Some new results on neural net approximation. *Neural Networks*, **6**:1069–1072, 1993.

[63] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**:359–366, 1989. Reprinted in [157].

[64] Hornik, K., Stinchcombe, M., White, H., and Auer, P. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. *Neural Computation*, **6**:1262–1275, 1994.

[65] Hush, D. R. and Horne, B. G. Progress in supervised neural networks: What's new since Lippmann. *IEEE Signal Processing Magazine*, **10**(1):8–39, January 1993.

[66] Hwang, J. N., Lay, S. R., Maechler, M., Martin, R. D., and Schimert, J. Regression modelling in back-propagation and projection pursuit learning. *IEEE Transactions on Neural Networks*, **5**(3):342–353, 1994.

[67] Ienne, P. and Kuhn, G. Digital systems for neural networks. In Papamichalis, P. and Kerwin, R., editors, *Digital Signal Processing Technology*, volume CR57 of *Critical Reviews Seires*, pages 314–345. SPIE Optical Engineering Press, Orlando, FL, 1995. Also available as `ftp://mantraftp.epfl.ch/mantra/-ienne.spie95.A4.ps.gz`.

[68] Ito, Y. Approximation of continuous functions on $\mathbb{R}^d$ by linear combinations of shifted rotations of a sigmoid function with and without scaling. *Neural Networks*, **5**:105–116, 1992.

[69] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science*, **220**:671–680, 1983.

[70] Kiselewich, S. J. and Turner, D. T. Using a neural network to distinguish between deployment events and non-deployment events in a supplemental inflatable

restraint system. *Automotive Engineering*, **103**(6):S5, June 1995. Related article available as `http://www.delco.com/techpapers/tech_neural.html`.

[71] Knowler, W. C., Bennet, P. H., Hamman, R. F., and Miller, M. Diabetes incidence and prevalence in Pima Indians: A 19-fold greater incidence than in Rochester, Minnesota. *American Journal of Epidemiology*, **108**:497–505, 1978.

[72] Knowler, W. C., Pettitt, D. J., Savage, P. J., and Bennet, P. H. Diabetes incidence in Pima Indians: Contributions of obesity and parental diabetes. *American Journal of Epidemiology*, **113**:144–156, 1981.

[73] Koistinen, P. and Holmstrom̈, L. Kernel regression and backpropagation training with noise. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, volume 4, pages 1033–1039. Morgan Kaufmann, San Mateo, CA, 1992.

[74] Kuan, C. M. and Hornik, K. Convergence of learning algorithms with constant learning rates. *IEEE Transactions on Neural Networks*, **2**(5):484–489, 1991.

[75] Kushner, H. Asymptotic global behavior for stochastic approximations and diffusions with slowly decreasing noise effects: Global minimisation via Monte Carlo. *SIAM Journal on Applied Mathematics*, **47**:169–185, 1987.

[76] Kwan, H. K. and Tang, C. Z. Designing multilayer feedforward neural networks using simplified sigmoid activation functions and one-power-of-two weights. *Electronics Letters*, **28**(25):2343–2344, December 1992.

[77] Kwan, H. K. and Tang, C. Z. Multiplierless multilayer feedforward neural network design suitable for continuous input-output mapping. *Electronics Letters*, **29**(14):1259–1260, July 1993.

[78] Lang, K. J. and Witbrock, M. J. Learning to tell two spirals apart. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, Pittsburgh 1988, 1989. Morgan Kaufmann, San Mateo, CA.

[79] Le Cun, Y. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

[80] Le Cun, Y. *Modeles connexionistes de l'apprentissage*. PhD thesis, Université Peierre et Marie Curie, 1987.

[81] Le Cun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, Denver 1989, 1990. Morgan Kaufmann, San Mateo, CA.

[82] Le Cun, Y., Kanter, I., and Sola, S. A. Eigenvalues of covariance matrices: Application to neural network learning. *Physical Review Letters*, **66**(18):2396–2399, 1991.

[83] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. Multilayer feedforward networks with nonpolynomial activation functions can approximate any function. *Neural Networks*, **6**:861–867, 1993.

[84] Lisboa, P. G. J. Private communication, September 1996.

[85] Lisboa, P. G. J. and Taylor, J. G., editors. *Workshop on Neural Network Applications and Tools*, Liverpool, England, September 1993, 1994. IEEE Computer Society Press, Los Alamitos, CA.

[86] MacKay, D. A practical Bayesian framework for backpropagation networks. *Neural Computation*, **4**(3):448–472, 1992.

[87] MacKay, D. J. Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks. Technical report, Cavendish Laboratory, University of Cambridge, Cambridge, England, 1995. Available as `ftp://mraos.ra.phy.cam.ac.uk/pub/mackay/network.ps.Z`.

[88] Marchesi, M., Benvenuto, N., Orlandi, G., Piazza, F., and Uncini, A. Design of multi-layer neural networks with power-of-two weights. In *IEEE ISCS*, volume 4, pages 2951–2954, New Orleans 1-3 May 1990, 1990. IEEE Press, New York, NY.

[89] Marchesi, M., Orlandi, G., Piazza, F., and Uncini, A. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, **4**(1):53–62, 1993.

[90] Marsden, J. E. *Elementary Classical Analysis*. W. H. Freeman, San Francisco, 1974.

[91] Masters, T. *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, chapter 3. Wiley, New York, NY, 1994.

[92] Mauduit, N., Duaranton, M., Gobert, J., and Sirat, J. A. L Neuro 1.0: A piece of hardware LEGO for building neural network systems. *IEEE Transactions on Neural Networks*, **3**(3):414–422, 1992.

[93] Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, 1994.

[94] Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors. *Machine Learning, Neural and Statistical Classification*, chapter 6. In Michie et al. [93], 1994.

[95] Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors. *Machine Learning, Neural and Statistical Classification*, chapter 7. In Michie et al. [93], 1994.

[96] Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors. *Machine Learning, Neural and Statistical Classification*, chapter 9. In Michie et al. [93], 1994.

[97] Moody, J. E. The effective number of parameters: An analysis of generalisation and regularization in nonlinear learning systems. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann, San Mateo, CA, 1992.

[98] Murphy, P. M. and Aha, D. W. UCI Repository of Machine Learning. Department of Information and Computer Science, University of California, Irvine, CA, 1995. The repository is accessible as `ftp://ics.uci.edu/pub/-machine-learning-databases/`.

[99] Murray, A. F. Multilayer perceptron learning optimised for on-chip implementation – A noise robust system. *Neural Computation*, **4**(3):366–381, 1992.

[100] Murray, A. F. and Edwards, P. J. Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvements. *IEEE Transactions on Neural Networks*, **4**(4):722–725, 1993.

[101] Mustafa, T. Private communication, June 1994.

[102] Neal, R. M. *Bayesian Learning in Neural Networks*. PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[103] Nelson, M. C. and Illingworth, W. T. *A Practical Guide to Neural Nets*. Addison-Wesley, Reading, MA, 1991.

[104] Nilsson, N. *Learning Machines: Foundations of Trainable Pattern Classifiers*. McGraw Hill, New York, NY, 1965. Republished as *The Mathematical Foundations of Learning Machines*, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[105] Nowlan, S. J. and Hinton, G. E. Simplifying neural networks by soft weight-sharing. *Neural Computation*, **4**(4):473–493, 1992.

[106] Obradovic, Z. and Parberry, I. Computing with discrete multi-valued neurons. *Journal of Computer and System Sciences*, **45**:471–492, 1992.

[107] Oja, E. Position paper before panel discussion on Neural Networks and Statistical Models. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 20–21, Washington, DC, June 1996. IEEE Press, New York, NY.

[108] Oteki, S., Hashimoto, A., Furuta, T., Watanabe, T., Stork, D. G., and Eguchi, H. A digital neural network vlsi with on-chip learning using stochastic pulse encoding. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 3039–3045, Nagoya, Japan, October 1993. IEEE Press, New York, NY.

[109] Parker, D. B. Learning-logic: Casting the cortex of the human brain in silicon. Technical report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, April 1985.

[110] Prechelt, L. Adaptive parameter pruning in neural networks. Technical report TR-95-009, International Computer Science Institute, Berkeley, CA, March 1995.

[111] *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, June 1996. IEEE Press, New York, NY.

[112] Reyneri, L. M. and Filippi, E. An analysis on the performance of silicon implementations of backpropagation algorithms for artificial neural networks. *IEEE Transactions on Computers*, **40**(12):1380–1389, 1991.

[113] Ripley, B. D. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, England, 1996.

[114] Ripley, B. D. *Pattern Recognition and Neural Networks*, chapter 1. In [113], 1996.

[115] Ripley, B. D. *Pattern Recognition and Neural Networks*, chapter 5. In [113], 1996.

[116] Ripley, B. D. *Pattern Recognition and Neural Networks*, chapter 6. In [113], 1996.

[117] Ripley, B. D. *Pattern Recognition and Neural Networks.* In [113], 1996. Glossary.

[118] Robbins, H. and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics*, **22**:400–407, 1951.

[119] Rudin, W. *Principles of Mathematical Analysis*, chapter 7. McGraw-Hill, New York, 1976.

[120] Rudin, W. *Principles of Mathematical Analysis*, chapter 8. In [119], 1976. Theorem 8.4.

[121] Rudin, W. *Principles of Mathematical Analysis*, chapter 8. In [119], 1976. Corollary to Theorem 8.1.

[122] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In Rumelhart et al. [123], chapter 8, pages 318–362.

[123] Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[124] Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, MA, 1986.

[125] Sarle, W. S. Subject: How can generalization error be estimated? Neural Network FAQ, part 2 of 7: Learning. Available as `ftp://ftp.sas.com/pub/neural/FAQ2.html`.

[126] Sarle, W. S. Subject: What is backprop? Neural Network FAQ, part 2 of 7: Learning. Available as `ftp://ftp.sas.com/pub/neural/FAQ2.html`.

[127] Sarle, W. S. Neural network and statistical models. In *Proceedings of the 19th Annual SAS Users Group International Conference, Cary, NC*, April 1994. Also available as `ftp://ftp.sas.com/pub/sugi19/neural/neural1.ps`.

[128] Sarle, W. S. Neural network implementation in SAS software. In *Proceedings of the 19th Annual SAS Users Group International Conference, Cary, NC*, April 1994. Also available as `ftp://ftp.sas.com/pub/sugi19/neural/neural2.ps`.

[129] Sarle, W. S. Re: Preprocessing data for ANN. Article 17388 of comp.ai.neural-nets newsgroup, August 1994.

[130] Sarle, W. S. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on Interface*, 1995. Also available as `ftp://ftp.sas.com/pub/neural/inter95.ps.Z`.

[131] Scaletter, R. and Zee, A. Emergence of grandmother memory in feedforward networks: Learning with noise and forgetfulness. In Waltz, D. and Feldman, J. A., editors, *Connectionist Models and Their Implications: Readings from Cognitive Science*, pages 309–332. Ableex, Norwood, MA, 1988.

[132] Sejnowski, T. J. and Rosenberg, C. R. NETtalk: A parallel network that learns to read aloud. Technical report JHU/EECS-86/01, Department of Electrical Engineering and Computer Science, John Hopkins University, Baltimore, MD, 1986.

[133] Simmons, G. F. *Introduction to Topology and Modern Analysis*. McGraw-Hill, New York, 1963.

[134] Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., and Johannes, R. S. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988. The diabetes dataset is available in the directory `ftp://ics.uci.edu/pub/-machine-learning-databases/pima-indians-diabetes/`.

[135] Staib, W. E. and Staib, R. B. The intelligent arc furnace controller: A neural network electrode position optimization system for electric arc furnaces. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE Press, New York, NY, 1992.

[136] Stinchcombe, M. and White, H. Universal approximation using multilayer feedforward networks with non-sigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 613–617, Washington, DC, 1989. IEEE Press, New York, NY. Reprinted in [157].

[137] Stinchcombe, M. and White, H. Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 7–16, San Diego, CA, 1990. IEEE Press, New York, NY. Reprinted in [157].

[138] Tang, C. Z. and Kwan, H. K. Multilayer feedforward neural networks with single power-of-two weights. *IEEE Transactions on Signal Processing*, **41**(8):2724–2727, 1993.

[139] Tesauro, G., He, Y., and Ahmad, S. Asymptotic convergence of backpropagation. *Neural Computation*, **1**(3):382–391, 1989.

[140] Thodberg, H. H. Private communication, July 1996.

[141] Thodberg, H. H. A review of Bayesian neural network with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks*, **6**(1):56–72, 1996.

[142] Thurn, S. B. Backpropagation on the MONK's problems. In Thurn et al. [143], chapter 9.

[143] Thurn, S. B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., Jong, K. D., Dzeroski, S., Fahlman, S. E., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R. S., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., de Welde, W. V., Wenzel, W., Wnek, J., and Zhang, J., editors. *The MONK's problems: A performance comparison of different learning algorithms.* Technical reportCMU-CS-91-197. Carnegie Mellon University, Pittsburgh, PA, December 1990. Also available as `ftp://ics.uci.edu/pub/-machine-learning-databases/monks-problems/thrun.comparison.ps.Z`.

[144] Thurn, S. B., Mitchell, T., and Cheng, J. The MONK's comparision of learning algorithms – Introduction and survey. In Thurn et al. [143], chapter 1.

[145] Turney, P. D. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, **2**:369–409, March 1995. Also available as `http://www.cs.washington.edu/-research/jair/volume2/turney95a-html/title.html`.

[146] Unnikrishnan, K. P. and Venugopal, K. P. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, **6**:469–490, 1994.

[147] Venkatesh, S. S. Directed drift: A new linear threshold algorithm for learning binary weights on-line. *Journal of Computer and System Sciences*, **46**(2):198–217, 1993.

[148] Vincente, C. J. P., Carrabina, J., Girrado, F., and Valderrama, E. Learning algorithms for feed-forward neural networks with discrete synapses. In Prieto, A., editor, *Artificial Neural Networks, Proceedings of IWANN'91*, volume 540 of *Lecture Notes in Computer Sciences*, pages 144–152, Granada, Spain, September 1991. Springer–Verlag, Berlin, Germany.

[149] Wahaba, G. Generalization and regularization in nonlinear learning systems. In Arbib, M., editor, *The Handbook of Brain Theory and Neural Networks*, pages 426–430. MIT Press, Cambridge, MA, 1995.

[150] Wahba, G., Chong, G., Wang, Y., and Chappel, R. Soft classification, a.k.a risk estimation, via penalized log likelihood and smoothing spline analysis of variance. Technical report, Statistics Department, University of Wisconsin, Madison, WI, 1993.

[151] Weigend, A. On overfitting and the effective number of hidden units. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 335–342, 1994.

[152] Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, **1**(3):193–209, 1990.

[153] Weiss, S. M. and Kulikowski, C. A. *Computer Systems That Learn*, chapter 2. Morgan Kaufmann, San Mateo, CA, 1991.

[154] Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[155] White, H. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, **1**(4):425–464, 1989. Reprinted in [157].

[156] White, H. Some asymptotic results for learning in single hidden-layer feedforward networks. *Journal of the American Statistical Association*, **84**:1003–1013, 1989. (Correction: **87**, 1252.). Reprinted in [157].

[157] White, H. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Oxford, England, 1992.

[158] Whitehouse, D. J. and Huang, T. Adaptive control of electromagnetically levitated spindle using neural networks. EPSRC proposal, Department of Engineering, University of Warwick, Coventry, England, April 1996.

[159] Widrow, B., Rumelhart, D. E., and Lehr, M. A. Neural networks: Applications in industry, business and science. *Communications of the ACM*, **37**(3):93–105, March 1994.

[160] Wnek, J. *Hypothesis-driven Constructive Induction*. PhD thesis, George Mason University, March 1993. Also available as technical report MLI 93-2 of Learning and Inference Laboratory, Centre for Artificial Intelligence, School of Information Technology and Engineering.

[161] Wnek, J. and Michalski, R. S. Comparing symbolic and subsymbolic learning: Three studies. In Michalski, R. and Tecuci, G., editors, *Machine Learning: A Multistrategy Approach*, volume 4, pages 318–362. Morgan Kaufmann, San Mateo, CA, 1993.

[162] Wnek, J., Sarma, J., Wahab, A., and Michalski, R. Comparison learning paradigms via diagrammatic visulization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods. Technical report, Computer Science Department, George Mason University, 1990.

[163] Wolpert, D. H. Stacked generalization. *Neural Networks*, **5**(2):241–259, 1992.

[164] Woodland, P. C. Weight limiting, weight quantisation & generalisation in multilayer perceptrons. In *Proceedings IEE First International Conference Artificial Neural Nets*, pages 297–300, London, 1989. IEE, London, England.

[165] Wray, J. and Green, G. G. R. Neural networks, approximation theory, and finite precision computing. *Neural Networks*, **8**(1):31–37, 1995.

[166] Xie, Y. and Jabri, M. A. Training algorithms for limited precision feedforward neural nets. SEDAL technical report 1991-8-3, Department of Electrical Engineering, University of Sydney, NSW 2006, Australia, 1991.

[167] Xie, Y. and Jabri, M. A. Analysis of the effects of quantization on multilayer neural networks using a statistical model. *IEEE Transactions on Neural Networks*, **3**(2):334–338, 1992.

[168] Yasunaga, M., Masuda, N., Yagyu, M., Asai, M., Shibata, K., Ooyama, M., Yamada, M., Sakaguchi, T., and Hashimoto, M. A self-learning neural network composed of 1152 digital neurons in wafer scale LSIs. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1844–1849, Seattle, WA, July 1991. IEEE Press, New York, NY.

[169] Yoo, H. and Pimmel, R. L. Weight discretization in back-propagation neural network classifiers. In Dagli, C. H. et al., editor, *Intelligent Engineering Systems through Artificial Neural Networks*, pages 167–172. ASME Press, New York, 1991.

[170] Zhao, Q. and Tadokoro, Y. A simple design of FIR filters with powers-of-two coefficients. *IEEE Transactions on Circuits and Systems*, **CAS-35**:566–570, 1988.

A

Mathematical Proofs for Chapter 2

This appendix is an expanded version of Section 2.2.2. The added details include definitions, explanations, intermediate results, and proofs of the theorems and lemmas of Section 2.2.2

This appendix will follow closely the approach chosen by Stinchcombe and White [137], in which they enforced some restrictions on activation functions to achieve the universal approximation property for networks with bounded weights. The results of this appendix were used in the proof for multiplier-free network existence theorem of Section 5.2.1.

Consider the approximation of functions $f \in C(\mathbb{R}^d)$ with a 2-layer feedforward network $\mathcal{N}^d(\sigma, B)$, where σ is the continuous hidden layer activation function and B, $0 < B < \infty$, is the maximum allowed weight magnitude. Let $\mathbf{A}^d$ be the set of real **affine transforms**[1] on $\mathbb{R}^d$, $A : \mathbb{R}^d \to \mathbb{R}$, i.e. $A(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + \theta$, where $\mathbf{w} \in \mathbb{R}^d$, $\theta \in \mathbb{R}$, and $|A| = \max\{|w_i|, |\theta|\}$. A d-input feedforward network is defined as

$$\mathcal{N}^d(\sigma, B) = \left\{ \hat{f}(\mathbf{x}) = \sum_{j=1}^{q} w_j^o \sigma(A_j(\mathbf{x})) \colon A_j \in \mathbf{A}^d, \text{ and} \right.$$

[1] Rotations, shifts, and scalings, or any combination thereof, are affine transforms.

$$\max\left\{|w_j^o|, |A_j| : 1 \leq j \leq q\right\} \leq B\Bigg\}.$$

where, q is the number of hidden neurons, $\mathbf{x}$ is the d-dimensional input vector, $\mathbf{w}_j$ is the input layer synaptic vector for the jth hidden neuron, θ_j is the offset and $w_j^o \in \mathbb{R}$ the output layer synapse of that hidden neuron. This network is the same as that of Figure 1.3 except for the restriction on the magnitude of the weights.

A network $\mathcal{N}$ is defined to have the universal approximation property over $C(\mathbb{R}^d)$ if $\mathcal{N}$ is **dense**[2,3] in $C(\mathbb{R}^d)$. $\mathcal{N}^d(\sigma, B)$ is said to be **uniformly dense** in $C(\mathbb{R}^d)$ on the compact set $K \subset \mathbb{R}^d$ if for all $f \in C(\mathbb{R}^d)$ and every $\varepsilon > 0$ there exists $\hat{f} \in \mathcal{N}^d(\sigma, B)$ such that $\sup\{|f(\mathbf{x}) - \hat{f}(\mathbf{x})| : \mathbf{x} \in K\} < \varepsilon$. In this case, $\mathcal{N}^d(\sigma, B)$ is also uniformly dense in $C(K)$. $\mathcal{N}^d(\sigma, B)$ is **uniformly dense on compacta** in $C(\mathbb{R}^d)$ if it is uniformly dense in $C(\mathbb{R}^d)$ on every compact K. The goal of this appendix is to prove this 'uniform denseness on compacta' property for the CWN with bounded weights.

The following theorem can be used to simplify the investigation into the universal approximation property. It shows that the existence of a universal approximation proof in 1-dimension guarantees its extension in d-dimensions.

Theorem A.1 (Theorem 2.1) *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a Borel measurable function and let $0 < B < \infty$. If $\mathcal{N}^1(\sigma, B)$ is uniformly dense in $C(\mathbb{R})$ on some non-empty compact interval $[-s, s], s > 0$, then for $d \in \mathbb{N}$, $\mathcal{N}^d(\sigma, B)$ is uniformly dense on compacta in $C(\mathbb{R}^d)$.* □

The proof of Theorem A.1 requires the following theorem:

Theorem A.2 (Stone-Weierstrass Theorem) *Let $\mathcal{A}$ be an **algebra**[4] of real continuous functions on a compact set K. If $\mathcal{A}$ **separates points**[5] on K and if $\mathcal{A}$ **vanishes at no***

[2] A set A is dense in a set S if $A \subset S$ and $Cl(A) = S$.

[3] For example, rational numbers are dense in irrational numbers. That means between any two irrational numbers there is a rational one or alternatively any irrational number can be approximated within any desired tolerance with a rational number.

[4] A set of functions $\mathcal{A}$ is an algebra if $f, g \in \mathcal{A},\ \vartheta \in \mathbb{R} \Longrightarrow f + g \in \mathcal{A},\ f \cdot g \in \mathcal{A},$ and $\vartheta f \in \mathcal{A}$.

[5] A set of functions $\mathcal{A}$ separates points on a set S if for every $x, y \in S, x \neq y$, there exists $f \in \mathcal{A}$ such that $f(x) \neq f(y)$ [136].

point[6] *of K, then the uniform closure $\mathcal{B}$ of $\mathcal{A}$ consists of all real continuous functions on K (i.e. a function in $\mathcal{A}$ can approximate any real continuous function on K to any desired degree of accuracy) [136].* □

Proof of Theorem A.1 Let $K \subset \mathbb{R}^d$ be a compact set and let $f(x) = \sin x$. Let $\mathcal{F}$ be the set of finite linear combinations of elements of $\{f(A(\cdot)) : f \in C(\mathbb{R}),\ A \in \mathbf{A}^d\}$. $\mathcal{F}$ is an algebra that separates points and contains the constants. The Stone-Weierstrass theorem implies that $\mathcal{F}$ is uniformly dense on compacta in $C(\mathbb{R}^d)$. Thus it is sufficient to show that for every $A \in \mathbf{A}^d$, $f(A(\cdot))$ can be uniformly approximated on K by functions in $\mathcal{N}^d(\sigma, B)$. Pick $N > 0$ sufficiently large that $A(K) \subset [-s \cdot N, s \cdot N]$ and $|A|/N < B$. Note that the range of affine functions $A^* = A/N$ is contained in $[-s, s]$ and that $f(A(\mathbf{x})) = f(N \cdot A^*(\mathbf{x}))$. It is possible to pick $\hat{f} \in \mathcal{N}^d(\sigma, B)$ such that $\sup\{|\hat{f}(\lambda) - f(N \cdot \lambda)| : \lambda \in [-s, s]\} < \varepsilon$. This implies immediately that $\hat{f}(A^*(\cdot)) \in \mathcal{N}^d(\sigma, B)$ and $\sup\{|\hat{f}(A^*(\mathbf{x})) - f(A(\mathbf{x}))| : \mathbf{x} \in K\} < \varepsilon$. □

The main result of this appendix is that enforcing bounds on the weights does not destroy the universal approximation capability of a network as long as the activation function is **superanalytic** at some point with a positive radius of convergence. $\sigma \in C(\mathbb{R})$ is defined to be **analytic** at $a \in \mathbb{R}$ with a radius of convergence $r > 0$ if there is an infinite sequence of real numbers, $\{c_n\}$, $n \geq 0$, such that for $|x - a| < r$, $\sum_{n=0}^{\infty} c_n (x-a)^n$ converges and $\sigma(x) = \sum_{n=0}^{\infty} c_n (x-a)^n$. Furthermore, this analytic function σ is defined to be superanalytic at a with a radius of convergence r if for every $n \geq 1$, $c_n \neq 0$. By the next lemma, this superanalyticity property holds if σ is analytic at a with radius r and $c_n \neq 0$ for infinitely many n.

Lemma A.1 (Lemma 2.1) *If σ is analytic at $a \in \mathbb{R}$ with radius $r > 0$, $\sigma(\lambda) = \sum_{n=0}^{\infty} c_n (\lambda - a)^n$, $|\lambda - a| < r$, and $c_n \neq 0$ for infinitely many n then for every b in a dense subset of $(a - r, a + r)$, σ is superanalytic at b with radius of convergence*

[6] A set of functions $\mathcal{A}$ vanishes at no point of the set S if for each $x \in S$ there exists $f \in \mathcal{A}$ such that $f(x) \neq 0$ [136]. The set of all polynomials in one variable does have this property. The set of all odd polynomials does not have this property, say on $[-1, 1]$, since $f(-x) = -f(x)$ for every odd function f [119].

$s = \min\{b - (a - r),\ (a + r) - b\}$. □

Proof There is no loss of generality in setting $a = 0$. Rudin [120] shows that σ can be expanded in a power series about any b in $(-r, r)$. Specifically, for every $x \in (b - s, b + s)$, $\sigma(x) = \sum_{n=0}^{\infty} (n!)^{-1} \sigma^{(n)}(b)(x - b)^n$ where $\sigma^{(n)}(x)$ is the nth derivative of σ at x. Rudin [121] shows that $\sigma^{(k)}(x) = \sum_{n=k}^{\infty} n(n-1)\cdots(n-k+1)c_n x^{n-k}$. As σ is analytic at a with a with radius $r > 0$ and $c_n \neq 0$ for infinitely many n, $S(k) = \{x \in (-r, r) : \sigma^{(k)}(x) \neq 0\}$ is an open dense subset of $(-r, r)$. By **Baire's theorem**[7] the intersection of $S(k)$ is dense in $(-r, r)$ and the result follows. □

Some examples of functions that are superanalytic almost everywhere are sine, cosine, logistic, and hyperbolic tangent functions. Finite polynomials are examples of functions which are analytic but not superanalytic.

The statement of the main result:

Theorem A.3 (Theorem 2.2) *If for some $a \in \mathbb{R}$, $\sigma \in C(\mathbb{R})$ is superanalytic at a, with radius of convergence $r > 0$, then $\mathcal{N}^d(\sigma, B)$ is uniformly dense on compacta in $C(\mathbb{R}^d)$ for any $B \geq \max\{|a|, 1\}$.* □

For the proof of this theorem it will be assumed that $a = 0$. This is done by shifting the argument of σ and abusing notation. Thus, for every $-r < x < r$, $\sigma(x) = \sum_{\alpha=0}^{\infty} c_\alpha x^\alpha$. The proof makes use of the following lemma.

Lemma A.2 *If $\sigma \in C(\mathbb{R})$ is analytic at 0 with a radius of convergence $r > 0$, $\sigma(x) = \sum_{\alpha=0}^{\infty} c_\alpha x^\alpha$ for $|x| < r$, then for every $k > 0$ there is a sequence of functions $\{\hat{f}_{n,k} \in \mathcal{N}^1(\sigma, 1)\}$* ***converging uniformly***[8] *to $c_k x^k$ on compacta.* □

[7] Baire's theorem states that the intersection of any countable family of open dense sets in a locally compact space is dense [32]

[8] Uniform convergence is the property that all of a family of functions or series on a given set converge at the same rate throughout the set; that is, for every $\varepsilon > 0$ there is a single N such that for all points in the set , $|f_m(x) - f_n(x)| < \varepsilon$ for all $m, n > N$ and similarly for uniform convergence as x tends to a value a [20].

Proof A double array of functions $\{\hat{f}_{n,k} \in \mathcal{N}^1(\sigma,1) : n \geq 1, k \geq 0\}$ is constructed having the desired properties using functions $\{h_{n,k} \in \mathcal{N}^1(\sigma,1)\}$ such that $\{\hat{f}_{n,k} = n^k h_{n,k}(x/n)$ and $h_{n,k+1}(x) = h_{n,k}(x) - \hat{f}_{n,k}(x)$.

First thing to note is that if $h_{n,0} \in \mathcal{N}^1(\sigma,1)$, then $\hat{f}_{n,k} \in \mathcal{N}^1(\sigma,1),\ k \geq 0, n \geq 1$, because $n^k h_{n.k}(x/n) = \sum_{j=1}^{n^k} h_{n,k}(x/n)$ belongs to $\mathcal{N}^1(\sigma,1)$ if $h_{n,k}$ does. A general construction will be exhibited here such that $\hat{f}_{n,k} \to c_k x^k$ uniformly on compacta, and then it will be verified that $h_{n,0}$ can be selected to belong to $\mathcal{N}^1(\sigma,1)$. (In fact, $h_{n,0} = \sigma$.)

To obtain the desired construction let $\{q_{n,\alpha,0} : n \geq 1, \alpha \geq 0\}$ be a double array such that $0 \leq q_{n,\alpha,0} \leq 1$ for all integers $n \geq 1, \alpha \geq 0$, and for each $\alpha \geq 0,\ q_{n,\alpha,0} \uparrow 1$, i.e. $q_{n,\alpha,0}$ approaches 1 from bellow, as $n \to \infty$. For each $k \geq 1$ recursively define the double array $\{q_{n,\alpha,0} : n \geq 1, \alpha \geq k\}$ with $q_{n,\alpha,k+1} = q_{n,\alpha,k}(1 - n^k/n^\alpha),\ k \geq 0, \alpha \geq k+1$. Note that $0 \leq q_{n,\alpha,k} \leq 1$ for all integers $n \geq 1, \alpha \geq k$. Also, for each $\alpha \geq k+1,\ q_{n,\alpha,k} \uparrow 1$ as $n \to \infty$.

Because σ is analytic at zero with radius $r > 0$, for $x \in (-r,r)$ and $n \geq 1, k \geq 0$, a function $h_{n,k}(x)$ can be defined as $\sum_{\alpha=k}^{\infty} q_{n,\alpha,k} c_\alpha x^\alpha$; the behaviour of $h_{n,k}$ outside of $(-r,r)$ is of no concern. Putting $\hat{f}_{n,k}(x) = n^k h_{n,k}(x/n)$, it is clear that $\hat{f}_{n,k}$ is analytic at 0 with radius $(-nr, nr)$. Further, $h_{n,k+1}(x) = h_{n,k}(x) - \hat{f}_{n,k}(x)$ holds on $(-r,r)$ by choice of $\{q_{n,\alpha,k}\}$, as a direct calculation verifies.

Now for $x \in (-nr, nr)$,

$$\begin{aligned}\hat{f}_{n,k}(x) &= q_{n,k,k} c_k x^k + \sum_{\alpha=k+1}^{\infty} n^{k-\alpha} q_{n,\alpha,k} c_\alpha x^\alpha \\ &= q_{n,k,k} c_k x^k + \sum_{\alpha=1}^{\infty} n^{-\alpha} q_{n,\alpha+k,k} c_{\alpha+k} x^{\alpha+k}.\end{aligned}$$

Given a compact set $K \subset \mathbb{R}$, n can be chosen sufficiently large such that $K \subset (-nr, nr)$. Therefore, taking suprema over $x \in K$,

$$\begin{aligned}&\sup |\hat{f}_{n,k}(x) - c_k x^k| \ \leq \\ &\sup |q_{n,k,k} - 1| \cdot |c_k| \cdot |x|^k + \sup \sum_{\alpha=1}^{\infty} |q_{n,\alpha+k,k}| \cdot |c_{\alpha+k}| \cdot |x|^k \cdot |x/n|^\alpha.\end{aligned}$$

Because $q_{n,k,k} \uparrow 1,\ |c_k| < \infty$, and $x \in K$, the first term vanishes as $n \to \infty$. For

$\hat{f}_{n,k}(x) \to c_k x^k$ uniformly on K, it suffices that the second term vanish also. Now,

$$\sum_{\alpha=1}^{\infty} |q_{n,\alpha+k,k}| \cdot |c_{\alpha+k}| \cdot |x|^k \cdot |x/n|^\alpha \; =$$

$$|x|^{k+1} n^{-1} \sum_{\alpha=1}^{\infty} |q_{n,\alpha+k,k}| \cdot |c_{\alpha+k}| \cdot |x/n|^{\alpha-1}.$$

The analyticity of σ implies that $\sum_{\alpha=0}^{\infty} b_{n,\alpha} y^\alpha$, where $b_{n,\alpha} = |q_{n,\alpha+k+1,k}| \cdot |c_{\alpha+k+1}|$, defines a function analytic at 0 with radius r. Because $|x/n| < r$ for all $x \in K$ and n sufficiently large, $\sup_{x \in K} \sum_{\alpha=1}^{\infty} |q_{n,\alpha+k,k}| \cdot |c_{\alpha+k}| \cdot |x/n|^{\alpha-1} < \infty$. As $\sup_{x \in K} |x|^{k+1} < \infty$ and $n^{-1} \to 0$, the second term vanishes as required, so that $\hat{f}_{n,k}(x) \to c_k x^k$ uniformly on compacta. Finally, one may choose $h_{n,0} = \sigma$ by taking $q_{n,\alpha,0} = 1, \; n \geq 1, \alpha \geq 0$, thus ensuring that $\hat{f}_{n,k} \in \mathcal{N}^1(\sigma, 1)$. □

Proof of Theorem A.3 This follows immediately from the Stone-Weierstrass theorem using Lemma A.2 and the observations that $\mathcal{N}^1(\sigma, B)$ contains the constants, and that for every $n \geq 1, \; c_n \neq 0$. □

The activation functions are required to be superanalytic so that they can be represented as an infinite power series containing *all* powers of x. This 'infinite and complete polynomial' nature of these function makes them dense in all continuous functions.

The next theorem allows further restrictions on the weight values without sacrificing the universal approximation characteristic. The input layer synaptic vectors $\mathbf{w}_j$ can be confined to the unit sphere S^d at the expense of limiting the superanalytic activation functions to those whose derivatives form a basis for the continuous functions. This is due to the Stone-Weierstrass theorem which implies that the set of polynomials in σ – which in the present case is formed by the derivatives of σ – is uniformly dense in $C(\mathbb{R})$.

The following theorem uses the term **span** of a class of functions. For any function $f \in C(\mathbb{R})$ and $r > 0$, $f|(-r, r)$ denotes the restriction of f to the interval $(-r, r)$, and for any set of functions $\mathcal{F}$, $\mathcal{F}|(-r, r)$ denotes $\{f|(-r, r) : f \in \mathcal{F}\}$. For any $\mathcal{F}$ defined on a set $\mathcal{O}$, the span of $\mathcal{F}$, $sp(\mathcal{F})$, denotes the closure of the set of finite linear combinations of elements of $\mathcal{F}$ in the topology of uniform convergence on compact subsets of $\mathcal{O}$.

Theorem A.4 (Theorem 2.3) *If the conditions of Theorem A.3 hold and* $\mathbf{w}_j \in S^1 \cup \{0\}$ *then* $\mathcal{N}^1(\sigma, B)$ *is uniformly dense on compact subsets of* $(-r, r)$ *in* $sp(\{\sigma^{(k)}|(-r,r) : k \geq 0\})$ *for any* $B \geq 1$ *where* $\sigma^{(k)}$ *is the* k*-th derivative of* σ*. If in addition* $sp(\{\sigma^{(k)}|(-r,r) : k \geq 0\}) = C(\mathbb{R})|(-r,r)$ *then for* $\mathbf{w}_j \in S^d \cup \{0\}$, $\mathcal{N}^d(\sigma, B)$ *is uniformly dense on compacta in* $C(\mathbb{R}^d)$. □

The proof of this theorem requires some new terminology and a lemma. For every $f \in C(\mathbb{R})$, and $h_1 \neq 0$ define $\Delta_{h_1}^{(1)} f$ by $(\Delta_{h_1}^{(1)} f)(x) = h_1^{-1}(f(x + h_1) - f(x))$. Suppose that $\Delta_{h^*}^{(k-1)} f$ has been defined for every $h^* = (h_1, \ldots, h_{k-1}) \in \mathbb{R}^{k-1} \backslash \{0\}$. For $h = (h^*, h_k) \in \mathbb{R}^k \backslash \{0\}$ define $\Delta_h^{(k)} f$ by $(\Delta_h^{(k)} f)(x) = (\Delta_{h_k}^{(1)} \Delta_{h^*}^{(k-1)} f)(x)$.

Lemma A.3 *Let* $\sigma \in C(\mathbb{R})$ *have infinitely many derivatives on the set* $(-r, r)$*. For every* $k \geq 1$*, there is a sequence* $h^n \to 0$ *such that* $\Delta_{h^n}^{(k)} \sigma$ *converges to* $\sigma^{(k)}$ *uniformly on compact subsets of* $(-r, r)$ *as* $n \to \infty$. □

Proof It will be proved through induction. The result is true for $k = 1$ because σ has a continuous first derivative, and **pointwise convergence**[9] of continuous functions to a continuous function implies uniform convergence on compacta. Now it will be shown that the theorem is true for k based on the assumption that it is true for $k - 1$.

Let K be a compact subset of $(-r, r)$ and let $\varepsilon > 0$. It will be shown that there exists an $h = (h^*, h_k)$, $h^* \in \mathbb{R}^{k-1} \backslash \{0\}$, $h_k \neq 0$ such that

$$\sup \left\{ |(h_k)^{-1}[(\Delta_{h^*}^{(k-1)} \sigma)(x + h_k) - (\Delta_{h^*}^{(k-1)} \sigma)(x)] - \sigma^{(k)}(x)| : x \in K \right\} < \varepsilon.$$

Pick h_k such that $K \subset (-r + |h_k|, r - |h_k|)$ and

$$\sup \left\{ |(h_k)^{-1}[\sigma^{(k-1)}(x + h_k) - \sigma^{(k-1)}(x)] - \sigma^{(k)}(x)| : x \in K \right\} < \frac{\varepsilon}{3}.$$

This can be done because K is a compact subset of $(-r, r)$, σ has k continuous derivatives, and pointwise convergence of continuous functions to a continuous function implies uniform convergence on compacta. By the induction hypothesis it is possible to

[9] If $\{a_n\}$ is a sequence of non-random real variables then a_n converges to a, i.e., $a_n \to a$ as $n \to \infty$, if there exists a real number a such that for any $\epsilon > 0$, there exists an integer N_ϵ sufficiently large that $|a_n - a| < \epsilon$ for all $n \geq N_\epsilon$ [155]. Also known as deterministic convergence.

pick $h^* \in \mathbb{R}^{k-1} \backslash \{0\}$ such that

$$\sup \left\{ |(\Delta_{h^*}^{(k-1)} \sigma)(x) - \sigma^{(k-1)}(x)| : x \in [-r + |h_k|, r - |h_k|] \right\} < |h_k| \cdot \frac{\varepsilon}{3}.$$

Combining the last two inequalities and applying the triangle inequality completes the proof. □

Proof of Theorem A.4 The first part of the theorem follows from Lemma A.3 and the observation that $\Delta_h^{(k)} f \in \mathcal{N}^1(\sigma, B)$. The second part follows from Theorem A.1. □

Theorem A.4 states that the superanalyticity requirement for σ, although necessary, is not sufficient for universal approximation if synaptic vectors in the input layer are restricted to the unit-sphere. Consider the case of sine function, which is superanalytic everywhere except at 0, but the span of its derivatives is not dense, and therefore uniform approximation by $\mathcal{N}^d(\sigma, B)$ with input-layer synapses on the unit-sphere fails. This can be easily illustrated in the 1-dimension case. Let $\sigma(\cdot) = \sin(\cdot)$ and consider the set

$$\mathcal{N}^1(\sin, S^1) = \left\{ \hat{f} : \hat{f}(x) = \sum_{j=1}^{q} w_j^o \sin(\pm x + \theta_j),\ 1 \le j \le n,\ n \in \mathbb{N},\ \theta_j \in \mathbb{R} \right\}.$$

By the addition formulas for the sine function, any such $\hat{f}$ must be of the form $c_1 \sin(x) + c_2 \cos(x)$ for some $c_1, c_2 \in \mathbb{R}$. This set of functions, $\mathcal{N}^1(\sin, S^1)$, is clearly not dense on compacta in $C(\mathbb{R})$, being unable, for example, to universally approximate $\sin(20x)$ on $[-1, 1]$.

An application of Theorem A.4 provides the following very useful result:

Lemma A.4 (Lemma 2.2) *If σ is the logistic function then for $\mathbf{w}_j \in S^d \cup \{0\}$, $\mathcal{N}^d(\sigma, B)$ is uniformly dense in compacta in $C(\mathbb{R}^d)$.*

Proof For this it is sufficient to show that $sp(\{\sigma^{(k)} | (-r, r) : k \ge 0\}) = C(\mathbb{R}) | (-r, r)$. By the Stone-Weierstrass theorem it is known that for any continuous and strictly monotonic σ the set of polynomials in σ, $\left\{ \sum_k c_k \sigma^k : 0 \le k \le j \le n,\ n \in \mathbb{N} \right\}$ is uniformly dense in compacta in $C(\mathbb{R})$. Direct calculation shows that $\sigma^{(1)} = \sigma \cdot (1 - \sigma) = \sigma - \sigma^2$ and that the derivative of σ^k is $k \cdot \sigma^{k-1} \cdot \sigma \cdot (1 - \sigma) = k \cdot (\sigma^k - \sigma^{k+1})$. This implies

that the set of finite linear combinations of the derivative of σ is exactly the set of polynomials in σ. □

A similar application of Theorem A.4 was used in Chapter 5 to prove the universal approximation property of the multiplier-free net.

B

Training Procedure and Parameters

This appendix contains the details of the integer-weight learning procedure which was first presented in Table 3.3. It also displays the parameters values for all of the training runs whose results have been used in this thesis.

Table B.1 Abbreviations and symbols used in this appendix only[1]

C	Continuous weight net
E	Error controlling the values of χ and ρ. It can either be $E_{o_{rms}}$ or E_{o_m}
I	Integer weight net
M	Multiplier-free net
N	Network type
ε_n	Neuron error margin (Error in the output of a neuron is not backpropagated if it is within this margin [139])
k_η	BP learning rate boost factor (The learning rate is boosted by this factor for one epoch if $E_w = 0$ and $E_o > \varepsilon$
$k_{\sigma'}$	$\sigma'(\cdot)$ offset (Constant added to the derivative of the activation function if the error in the output of a neuron is large [35])
λ	Weight decay coefficient
-	Numerous values (Consult main text for the reason)

[1] See Page xi for the definitions of all other abbreviations and symbols.

Table B.2 Integer-weight learning[2]

procedure IntegerWeightLearning

repeat

initialise weights with values selected randomly

from a uniform distribution on the interval $(-0.5, 0.5)$;

repeat

$\chi \leftarrow \alpha_\chi e^{(\varepsilon - E_o)\beta_\chi}$;

$\rho \leftarrow \alpha_\rho e^{(\varepsilon - E_o)\beta_\rho}$;

shuffle training examples

repeat

choose next training pair $(\mathbf{x}, \mathbf{t})$; let the 0^{th} layer be $\mathbf{u}_0 = \mathbf{x}$;

ForwardPass;

ComputeGradients;

UpdateWeights;

until all training examples have been used

if $E_w = 0$ and $E_o > \varepsilon$ **then**

boost η by a factor of k_η for the next epoch only;

increment $epochs$;

until $(E_o < \varepsilon$ **and** $E_w = 0)$ **or** $(epochs$ **modulo** $C_R = 0)$;

until $epochs$ **modulo** $C_R \neq 0$

end; {IntegerWeightLearning}

subroutine ForwardPass

for $layer = 1$ **to** L **do**

for $neuron = 1$ **to** N_{layer} **do**

$u_{layer,neuron} \leftarrow \tanh\left(\sum_{i=0}^{N_{layer-1}} w_{layer,neuron,i} u_{layer-1,i}\right)$;

enddo

enddo

end; {ForwardPass}

[2]Notation and style adapted from [65].

(Continued on next page)

Table B.2 *(continued)*

subroutine ComputeGradients

for $layer = L$ **to** 1 **do**

for $neuron = 1$ **to** N_{layer} **do**

if $layer = L$ **then**

{

$e_{L,neurone} \leftarrow u_{L,neuron} - t_{neuron}$;

if $e_{L,neurone} < \varepsilon_n$ **then** $e_{L,neurone} \leftarrow 0$;

}

else

$e_{layer,neuron} \leftarrow$

$\sum_{m=1}^{N_{layer+1}} e_{layer+1,j}(1 - u_{layer+1,m}^2) w_{layer+1,m,neuron}$;

enddo

for all weights in layer $layer$ **do**

if $e_{layer,j} < 0.75$ **then**

$g_{layer,j,i} \leftarrow e_{layer,j}(1 - u_{layer,j}^2) u_{layer-1,i}$;

else

$g_{layer,j,i} \leftarrow e_{layer,j}(1 - u_{layer,j}^2 + k_{\sigma'}) u_{layer-1,i}$;

enddo

enddo

end; {ComputeGradients}

subroutine UpdateWeights

for all weights $w_{l,j,i}$ **do**

$w_{l,j,i} \leftarrow w_{l,j,i} - \eta g_{l,j,i} + \mu g_{l,j,i_{old}} - \lambda w_{l,j,i}$;

$w_{l,j,i} \leftarrow w_{l,j,i} - \chi \cdot (w_{l,j,i} - \mathcal{Q}_{prac}(w_{l,j,i})) \tan(RND)$;

$w_{l,j,i} \leftarrow \mathcal{B}(w_{l,j,i})$;

enddo

end; {UpdateWeights}

Table B.3 Training parameters for Chapter 3

Experiment		N	Confg	η	μ	λ	$k_{\sigma'}$	α_χ	α_ρ	E	k_η	ϵ	ϵ_n	C_R	C_T
Table 3.5	XOR	C	2:2:1	0.1	0.85	0	0.4	0	0	E_{o_m}	10	0.4	0.4	500	-
		I	2:2:1	0.1	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
	E/D4	C	4:2:4	0.05	0.9	0	0.4	0	0	E_{o_m}	10	0.4	0.4	500	-
		I	4:2:4	0.1	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
	E/D8	C	8:3:8	0.0075	0.9	0	0.4	0	0	E_{o_m}	10	0.4	0.4	500	-
		I	8:3:8	0.0075	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-

Table B.4 Training parameters for Chapter 4 and Appendix C

Experiment	N	Confg	η	μ	λ	$k_{\sigma'}$	α_χ	α_ρ	E	k_η	ϵ	ϵ_n	C_R	C_T
Figure 4.3	I	2:3:1	-	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	-	-	500	-
	I	2:4:1	-	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	-	-	500	-
Figure C.1	C	2:1	0.05	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:1	0.05	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.2	C	2:1	0.05	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:1	0.05	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.3	C	2:2:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:2:1	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.4	C	2:2:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:2:1	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.5	C	2:3:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:3:1	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.6	C	2:3:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	2:3:1	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure 4.7	C	2:4:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	1500	-
	I	2:4:1	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	1500	-

(Continued on next page)

Table B.4 *(continued)*

Experiment	N	Confg	η	μ	λ	$k_{\sigma'}$	α_χ	α_ρ	E	k_η	ϵ	ϵ_n	C_R	C_T
Figure 4.8	C	2:4:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	1500	-
	I	2:4:1	0.009	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	1500	-
Figure C.7	C	2:4:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	3000	-
	I	2:4:1	0.005	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	3000	-
Figure C.8	C	2:4:1	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	3000	-
	I	2:5:1	0.005	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	3000	-
Figure C.9	C	$\overrightarrow{2:1:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	$\overrightarrow{2:1:1}$	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.10	C	$\overrightarrow{2:1:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	$\overrightarrow{2:1:1}$	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.11	C	$\overrightarrow{2:1:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	$\overrightarrow{2:1:1}$	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure C.12	C	$\overrightarrow{2:1:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	500	-
	I	$\overrightarrow{2:1:1}$	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	500	-
Figure 4.9	C	$\overrightarrow{2:2:1}$	0.011	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	1500	-
	I	$\overrightarrow{2:2:1}$	0.011	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	1500	-
Figure 4.10	C	$\overrightarrow{2:2:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	1500	-
	I	$\overrightarrow{2:3:1}$	0.0075	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	1500	-
Figure C.13	C	$\overrightarrow{2:2:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	3000	-
	I	$\overrightarrow{2:4:1}$	0.009	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	3000	-
Figure C.14	C	$\overrightarrow{2:2:1}$	0.01	0.99	10^{-3}	0.4	0	0	E_{o_m}	0	0.4	0.4	3000	-
	I	$\overrightarrow{2:5:1}$	0.01	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	3000	-
Table 4.2 and	I	$\overrightarrow{2:1:1}$	0.01	0.99	0	0	10^{-3}	0.1	E_{o_m}	5	0.4	0.4	1000	-
Figure 4.6	I	$\overrightarrow{2:2:1}$	0.01	0.99	0	0	10^{-3}	0.1	E_{o_m}	5	0.4	0.4	1000	-

Table B.5 Training parameters for Chapter 5

Experiment		Confg	η	μ	λ	$k_{\sigma'}$	α_χ	α_ρ	E	k_η	ϵ	ϵ_n	C_R	C_T
Table 5.1	XOR	2:3:1	0.005	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	100	-
	E/D4	4:2:4	0.008	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	200	-
	E/D8	8:3:8	0.011	0.99	0	0.4	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	200	-

Table B.6 Training parameters for Chapter 6

Experiment	N		Confg	η	μ	λ	$k_{\sigma'}$	α_χ	α_ρ	E	k_η	ϵ	ϵ_n	C_R	C_T
Table 6.2	#1	I	17:4:1	0.05	0.15	0	0	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	200	171
		M	17:3:1	0.01	0.9	0	0.4	10^{-3}	0.1	$E_{o_{rms}}$	9.9	0.1	0.4	200	44
	#2	I	17:4*:1	0.1	0.075	0	0	10^{-3}	0.1	E_{o_m}	10	0.4	0.4	200	52
		M	17:3*:1	0.01	0.9	0	0.4	10^{-3}	0.1	$E_{o_{rms}}$	9.9	0.1	0.4	200	22
	#3	I	17:1	0.1	0.05	0	0.4	10^{-3}	0.1	$E_{o_{rms}}$	10	0.48	0.75	200	26
		M	17:7*:1	0.01	0.9	0	0.4	10^{-3}	0.1	$E_{o_{rms}}$	10	0.4	0.4	200	11
Table 6.4		C	8:2:1	0.005	0.8	10^{-4}	0	0	0	$E_{o_{rms}}$	0	0	0	50	590
		I	8:12*:1	0.005	0.8	0	0	10^{-3}	0.1	$E_{o_{rms}}$	10	0.75	0.4	50	84
		M	8:10*:1	0.005	0.8	0	0	10^{-3}	0.1	$E_{o_{rms}}$	10	0.75	0.4	50	183
Table 6.5		C	32:7:10	0.05	0.5	10^{-4}	0	0	0	$E_{o_{rms}}$	0	0.4	0.4	200	112
		I	32:10:10	0.025	0.8	0	0	10^{-3}	0.1	$E_{o_{rms}}$	1.5	0.05	0.4	200	3960
		M	32:11:10	0.0105	0.9	0	0.4	10^{-3}	0.1	$E_{o_{rms}}$	1.5	0.05	0.4	200	3500

*The number of hidden neurons is the one used at the start of simulations. Some of the hidden neurons were killed off during learning and the number of effective neurons is shown in the relevant table in the main text.

C

Decision Surfaces of Chapter 4

This appendix displays the decision surfaces mentioned but not shown in Chapter 4.

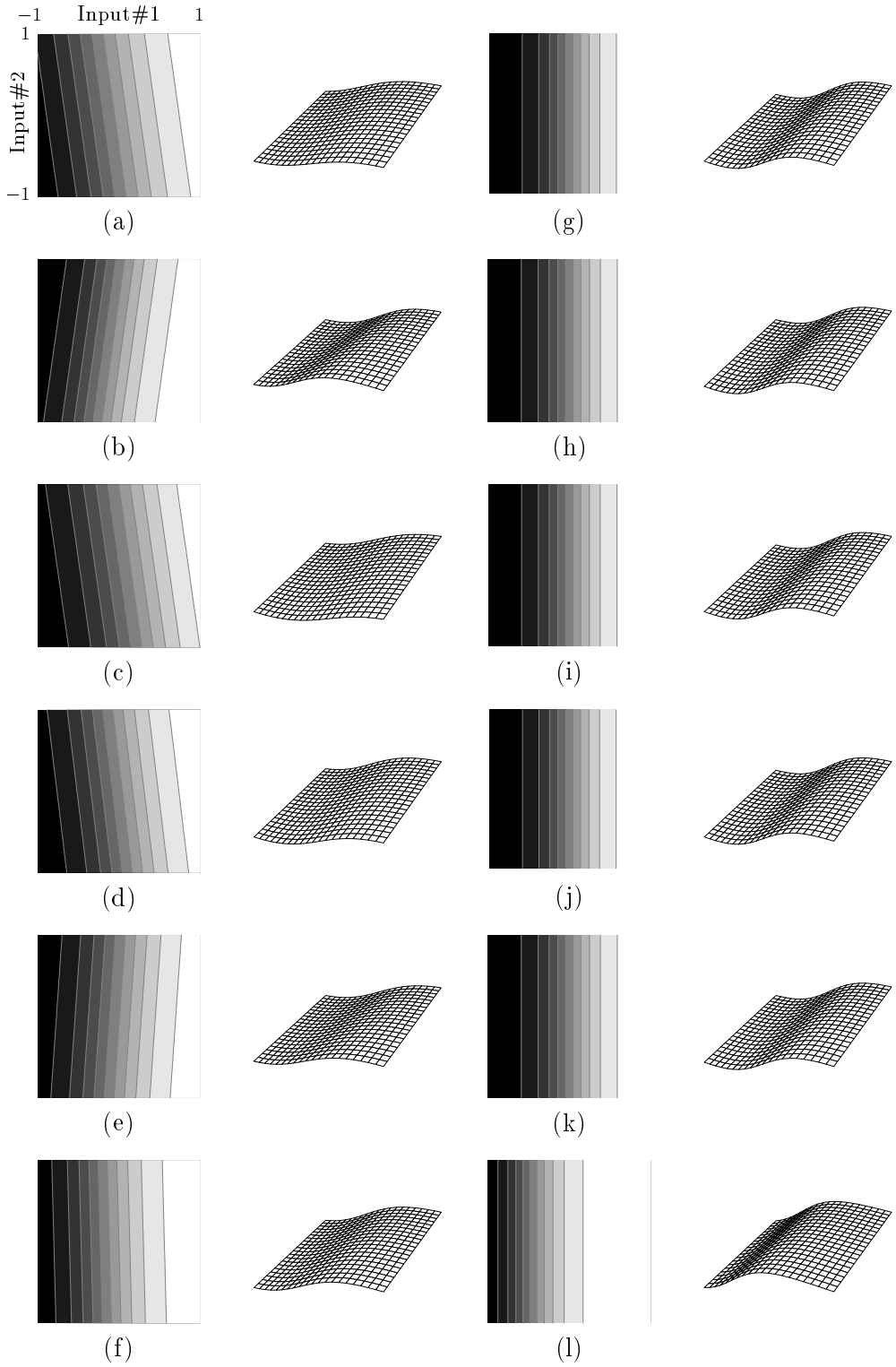


Figure C.1 Decision surfaces after 6 consecutive training runs on problem *A*: 2:1 network with double-precision weights (a-f); 2:1 network with integer weights (g-l).

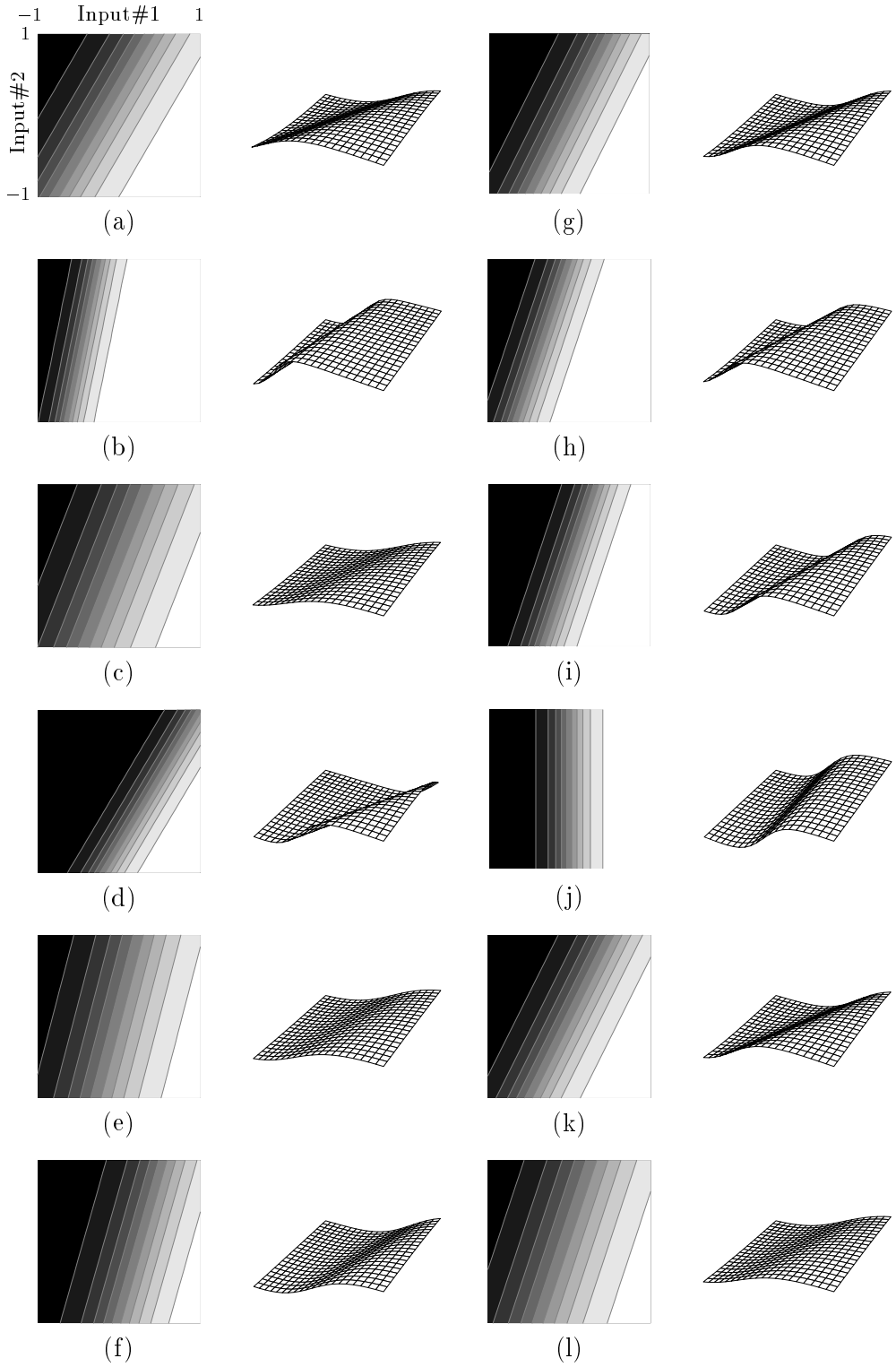


Figure C.2 Decision surfaces after 6 consecutive training runs on problem *A**: 2:1 network with double-precision weights (a-f); 2:1 network with integer weights (g-l).

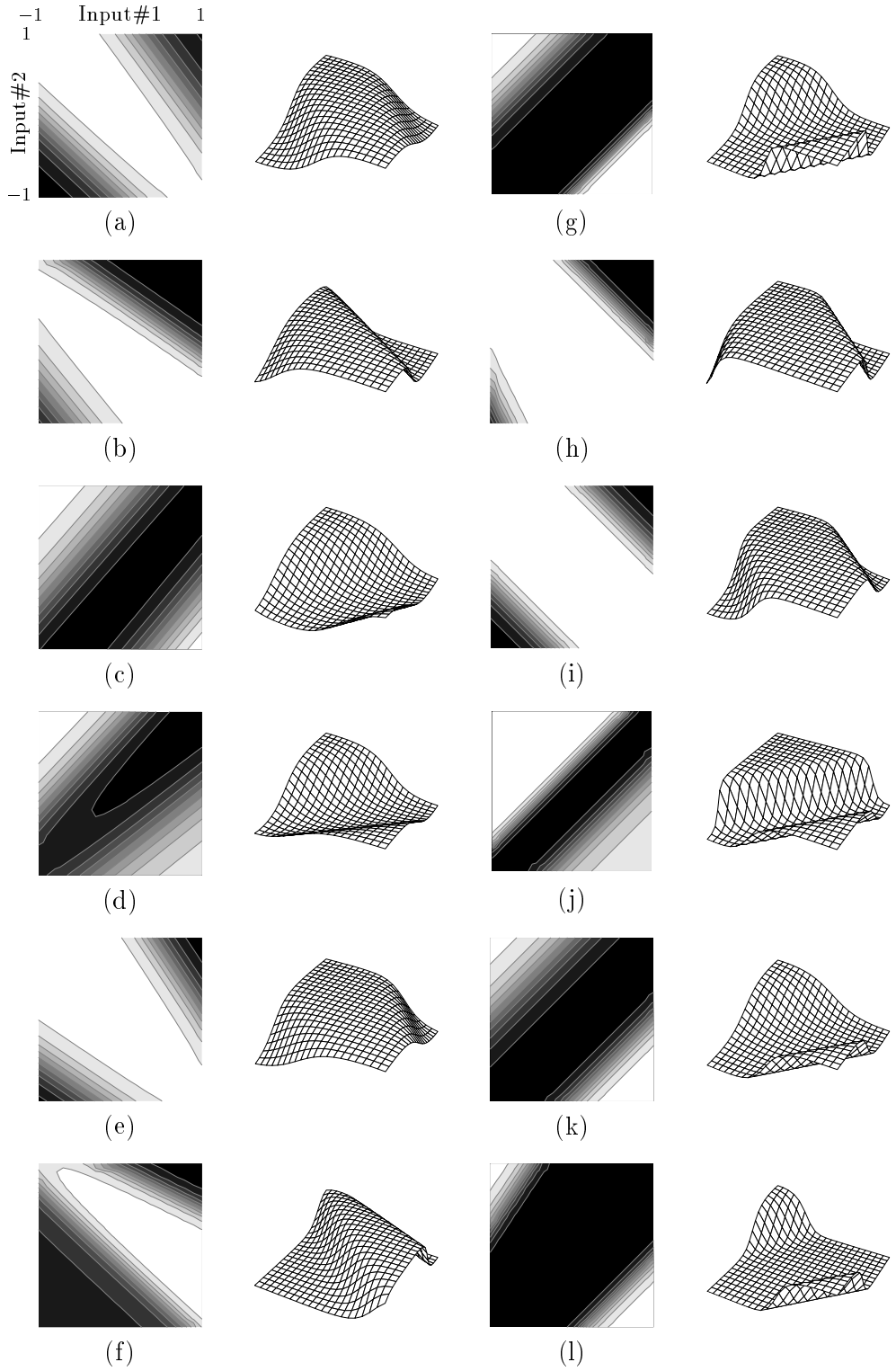


Figure C.3 Decision surfaces after 6 consecutive training runs on problem *B*: 2:1 network with double-precision weights (a-f); 2:1 network with integer weights (g-l).

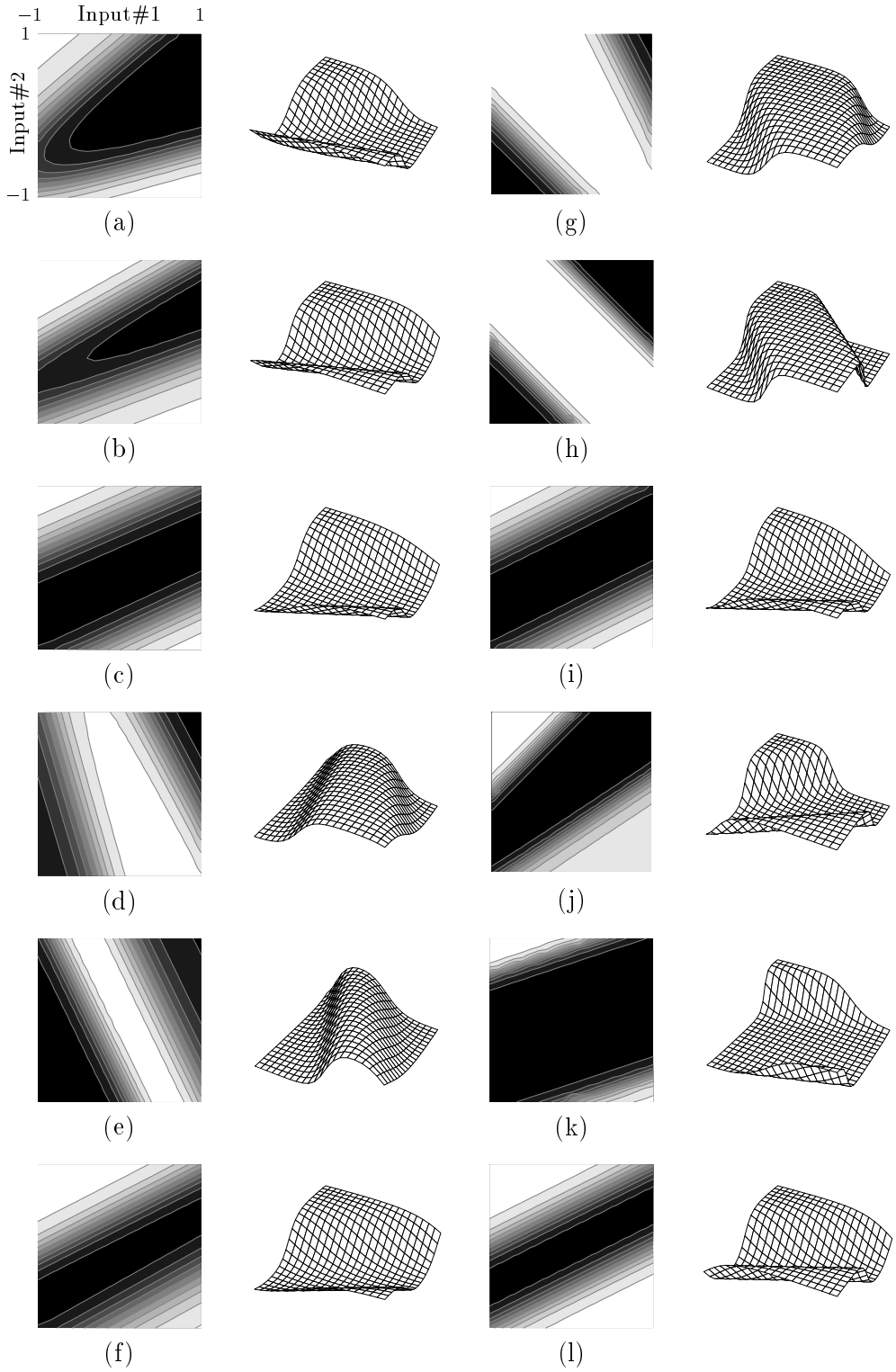


Figure C.4 Decision surfaces after 6 consecutive training runs on problem *B**: 2:2:1 network with double-precision weights (a-f); 2:2:1 network with integer weights (g-l).

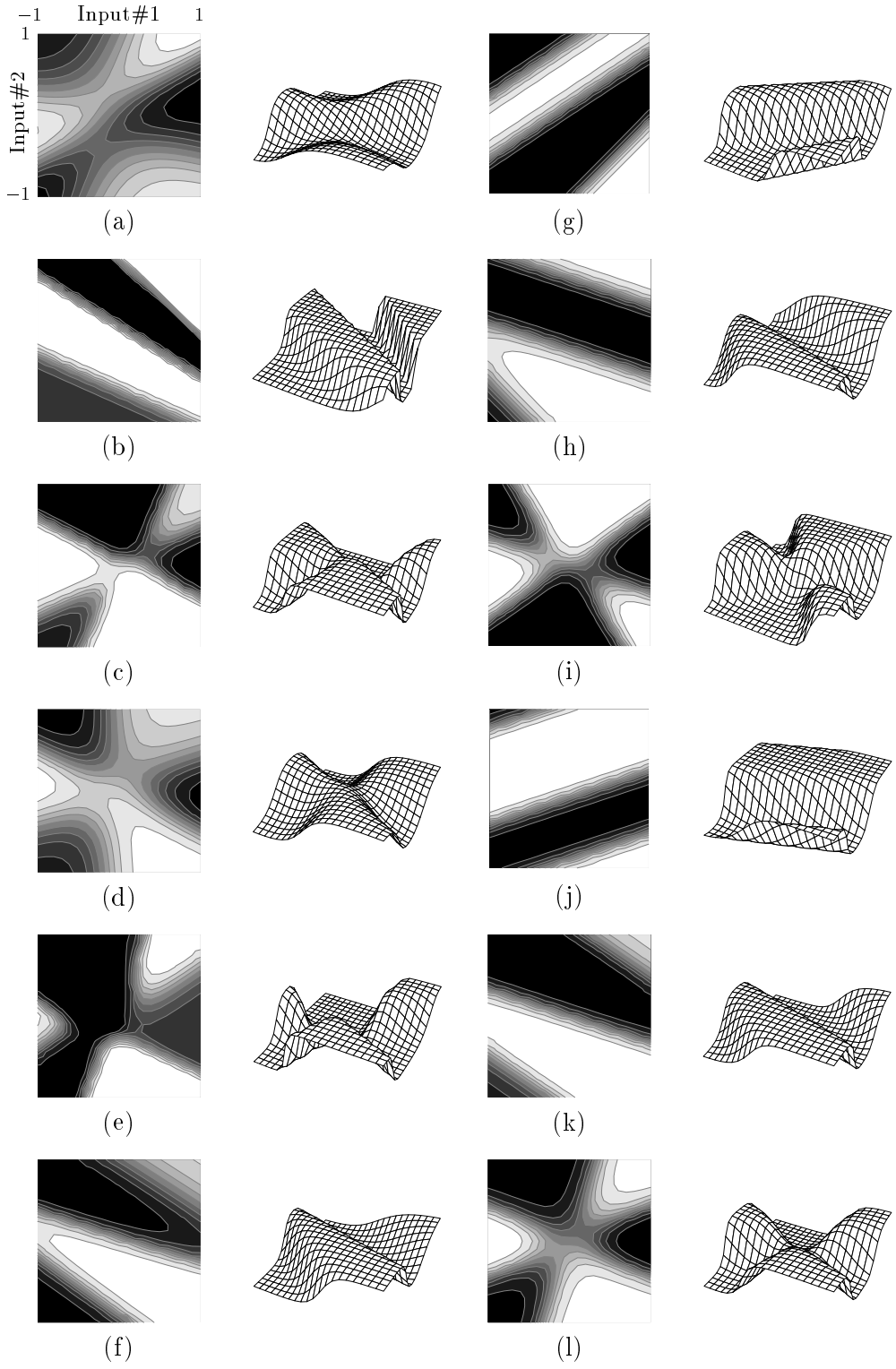


Figure C.5 Decision surfaces after 6 consecutive training runs on problem *C*: 2:3:1 network with double-precision weights (a-f); 2:3:1 network with integer weights (g-l).

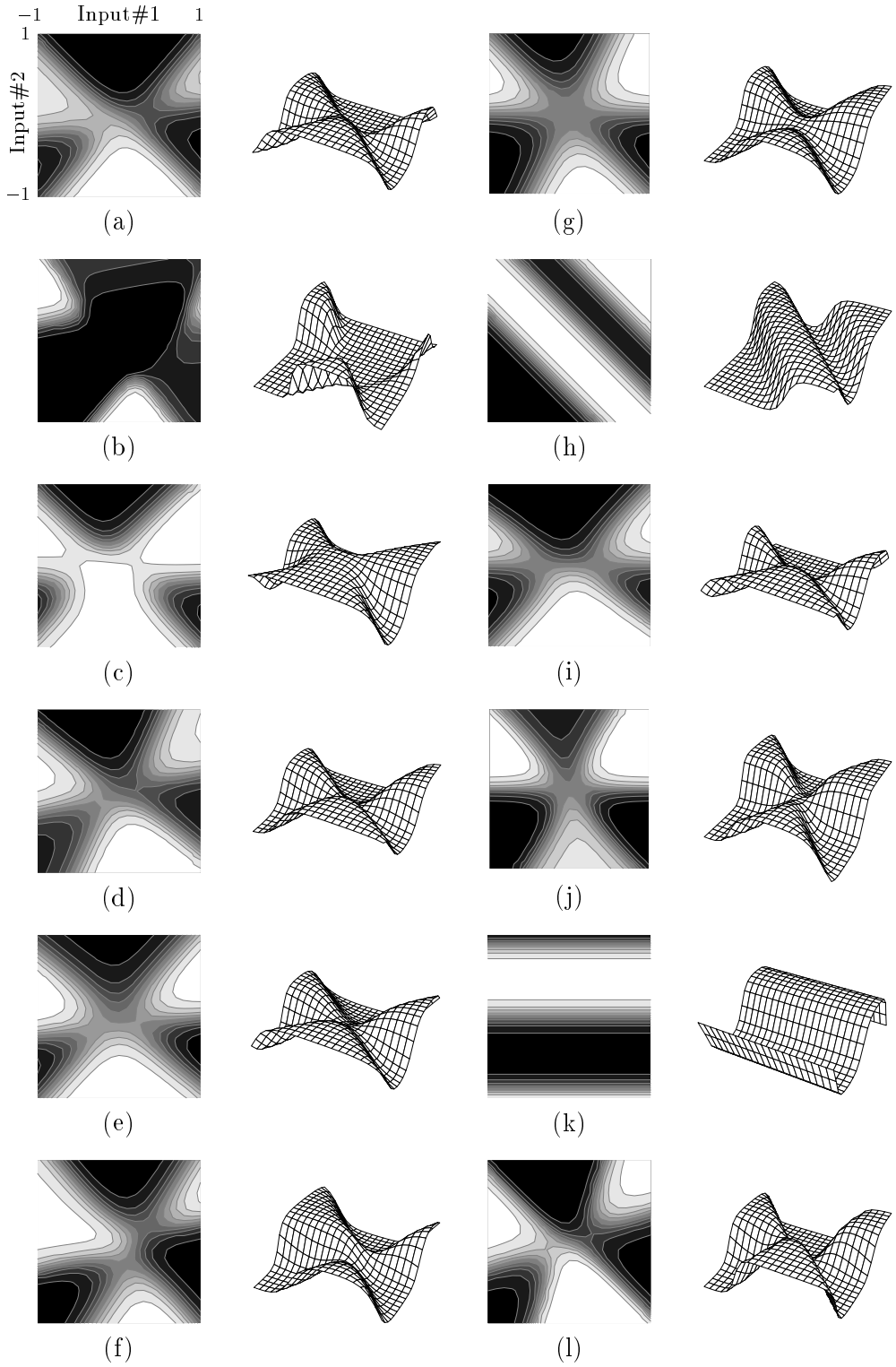


Figure C.6 Decision surfaces after 6 consecutive training runs on problem *C**: 2:3:1 network with double-precision weights (a-f); 2:3:1 network with integer weights (g-l).

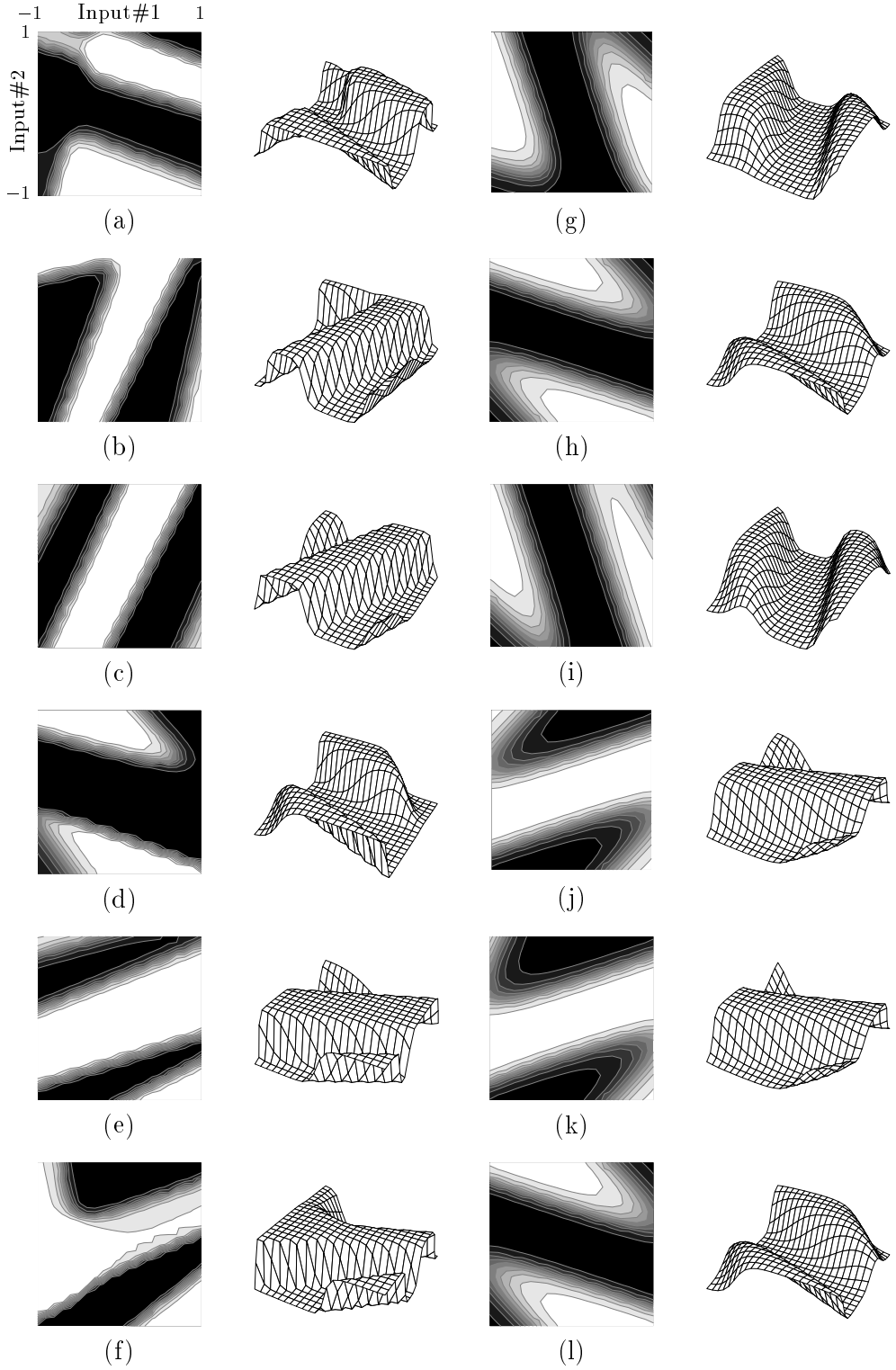


Figure C.7 Decision surfaces after 6 consecutive training runs on problem *E*: 2:4:1 network with double-precision weights (a-f); 2:4:1 network with integer weights (g-l).

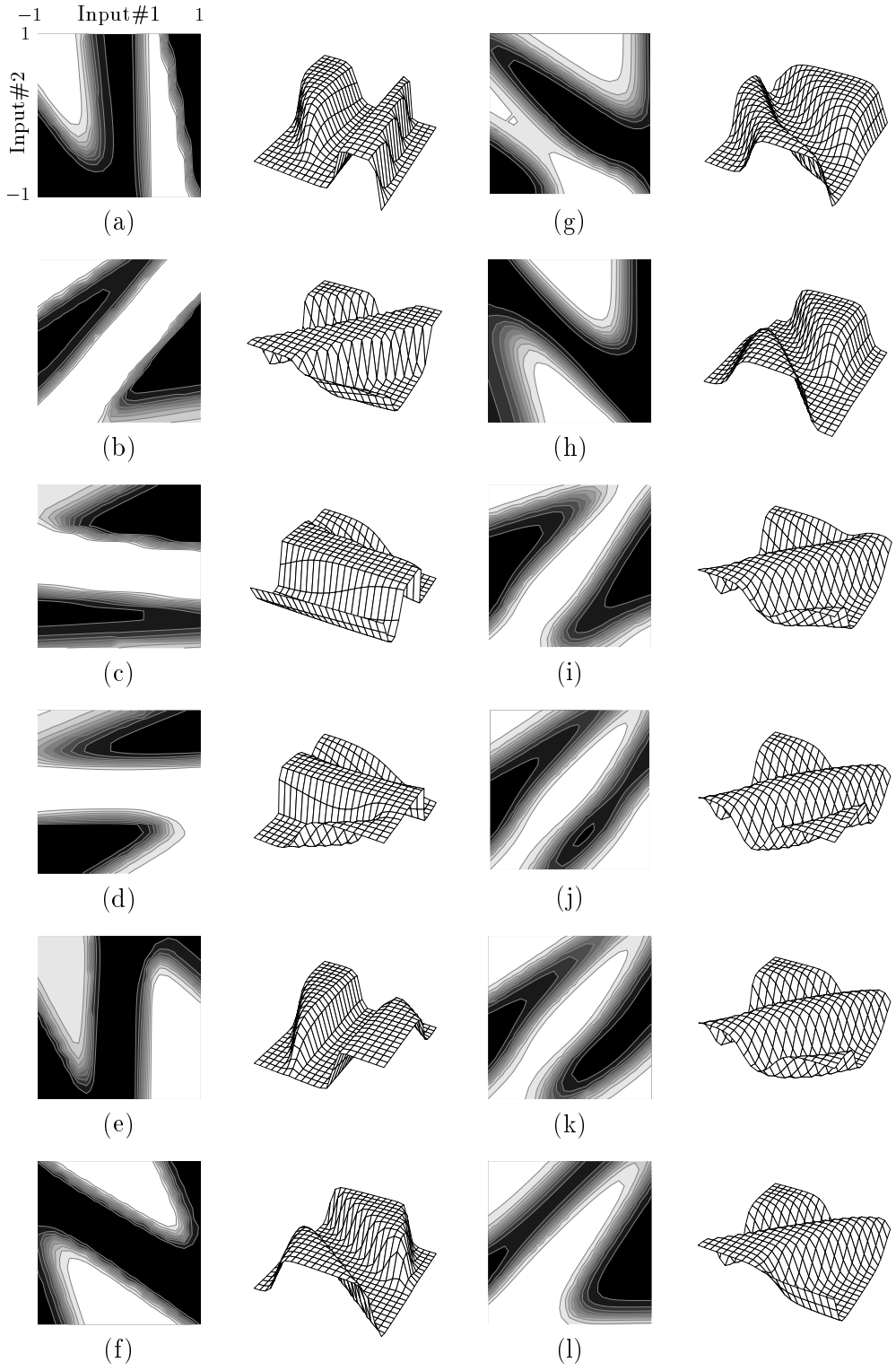


Figure C.8 Decision surfaces after 6 consecutive training runs on problem *E**: 2:4:1 network with double-precision weights (a-f); 2:5:1 network with integer weights (g-l).

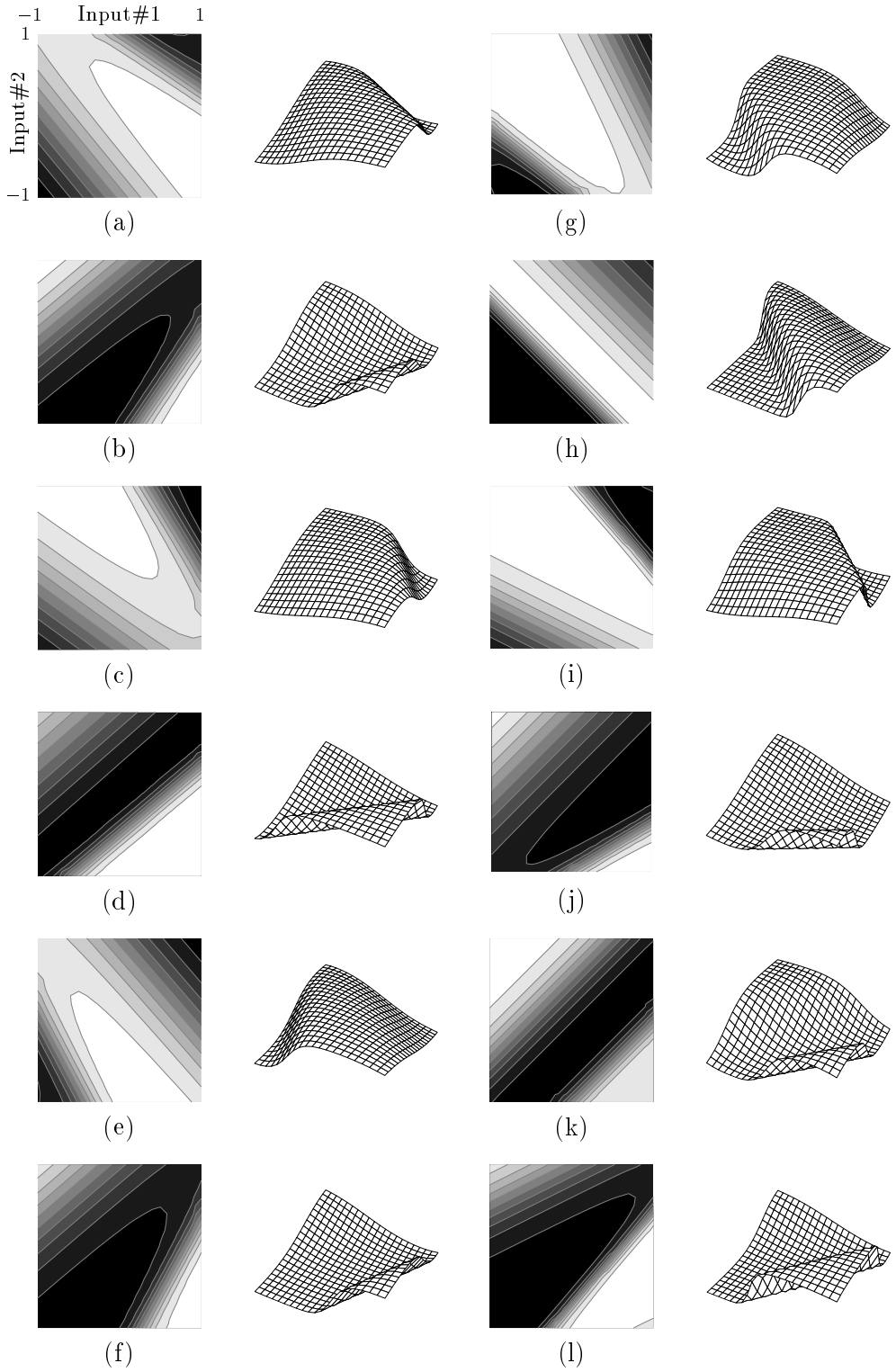


Figure C.9 Decision surfaces after 6 consecutive training runs on problem *B*: $\overrightarrow{2{:}1{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}1{:}1}$ network with integer weights (g-l).

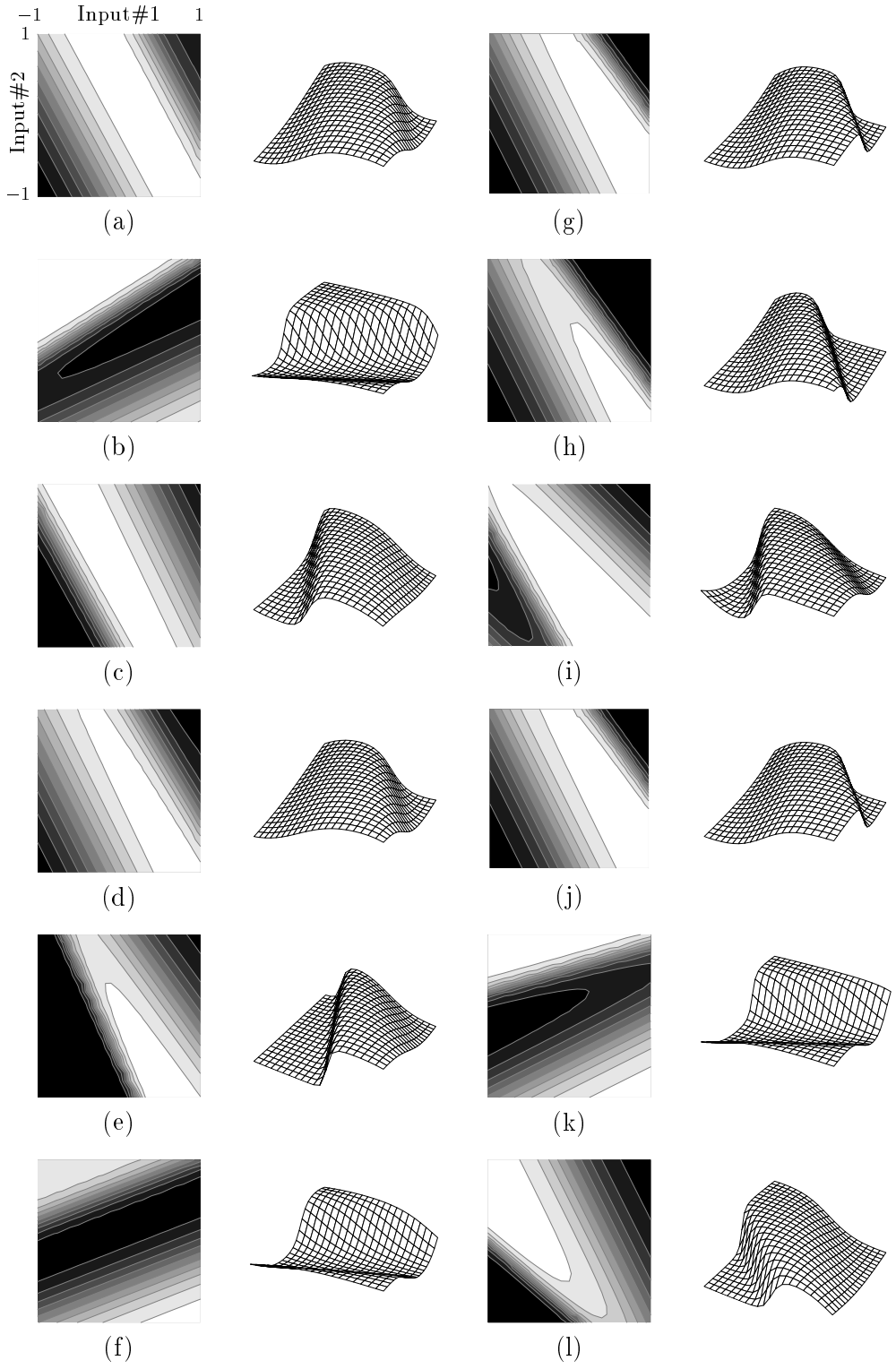


Figure C.10 Decision surfaces after 6 consecutive training runs on problem B^*: $\overrightarrow{2{:}1{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}1{:}1}$ network with integer weights (g-l).

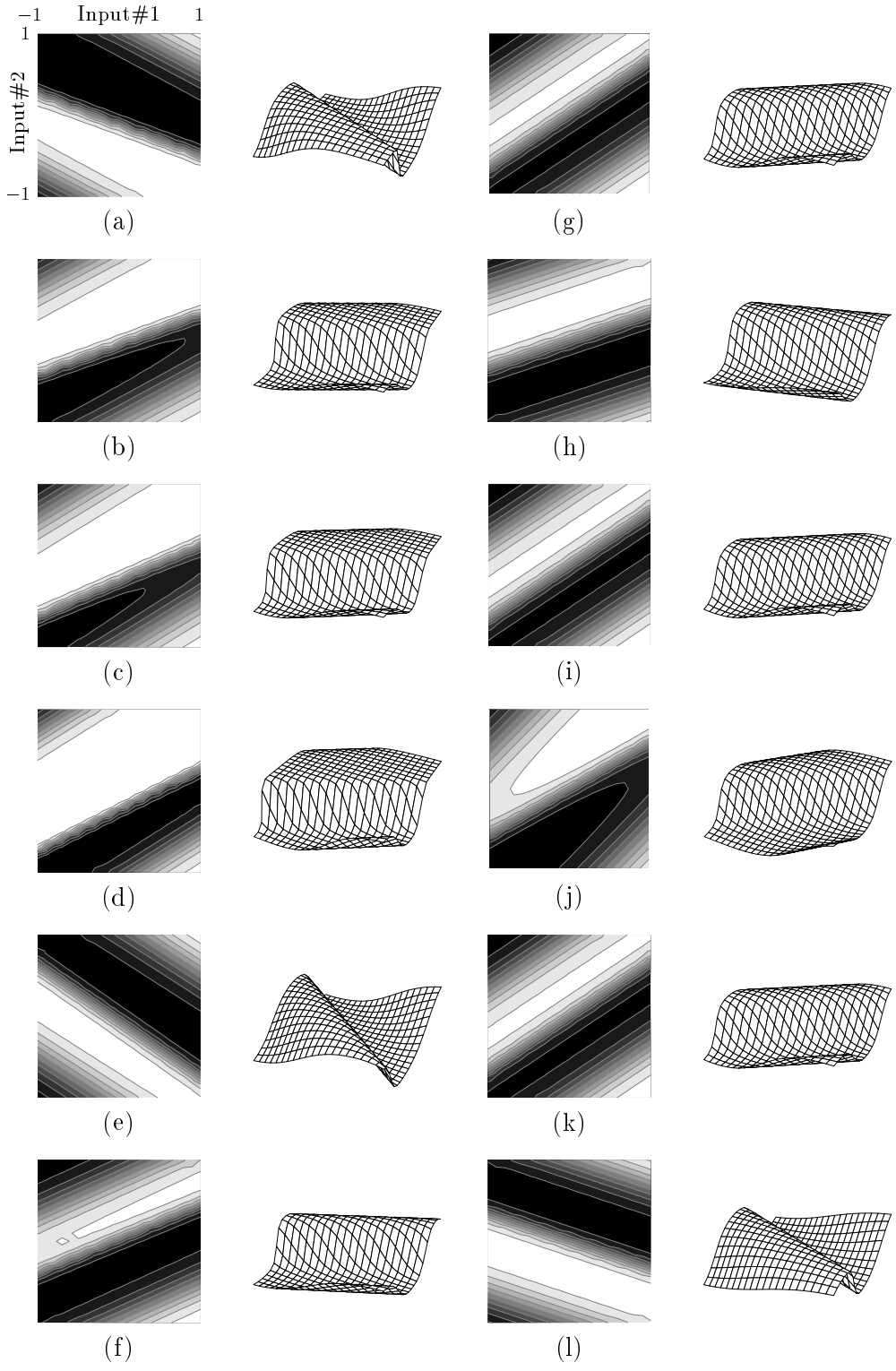


Figure C.11 Decision surfaces after 6 consecutive training runs on problem *C*: $\overrightarrow{2{:}1{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}1{:}1}$ network with integer weights (g-l).

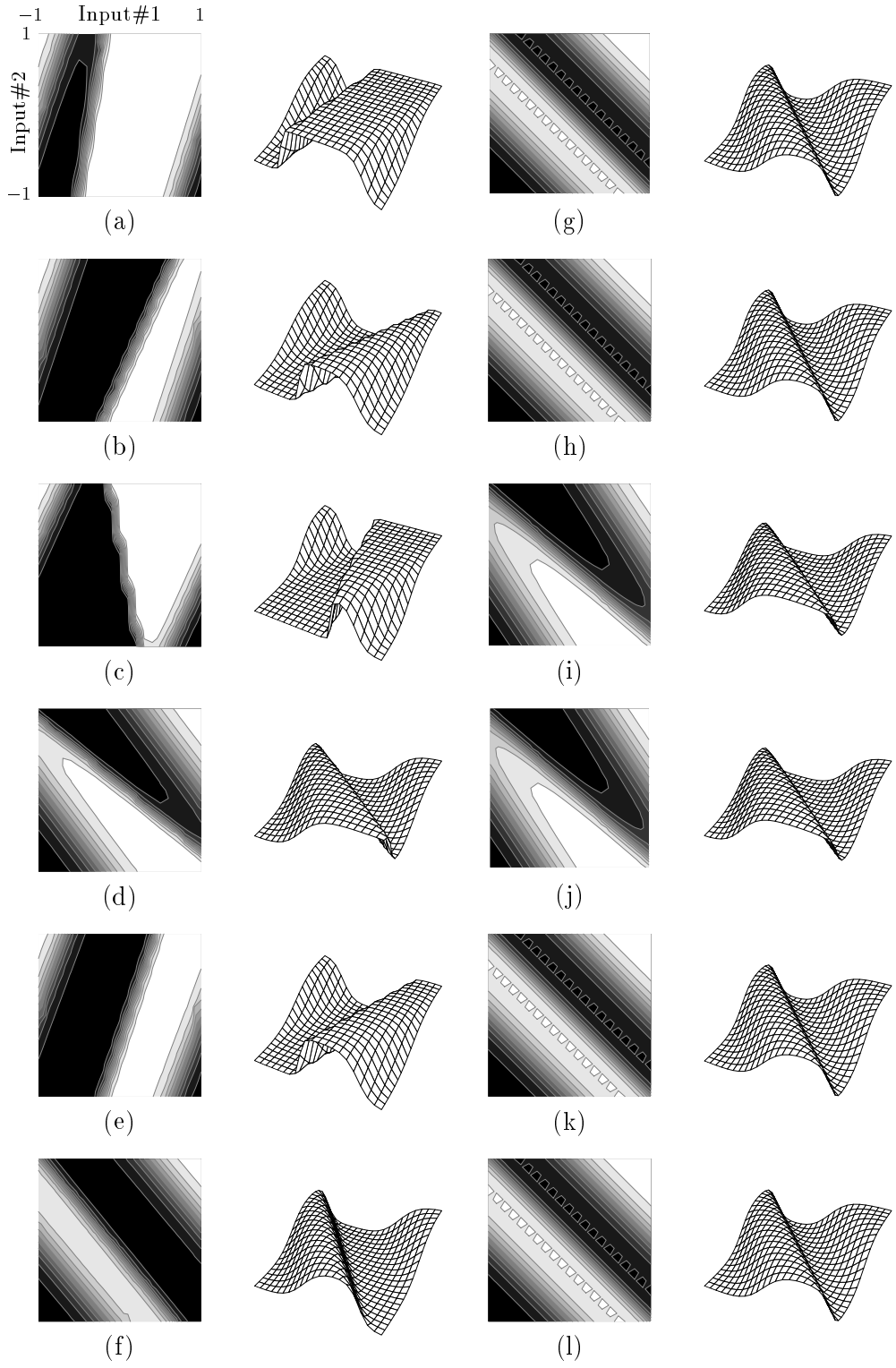


Figure C.12 Decision surfaces after 6 consecutive training runs on problem *C**: $\overrightarrow{2{:}1{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}1{:}1}$ network with integer weights (g-l).

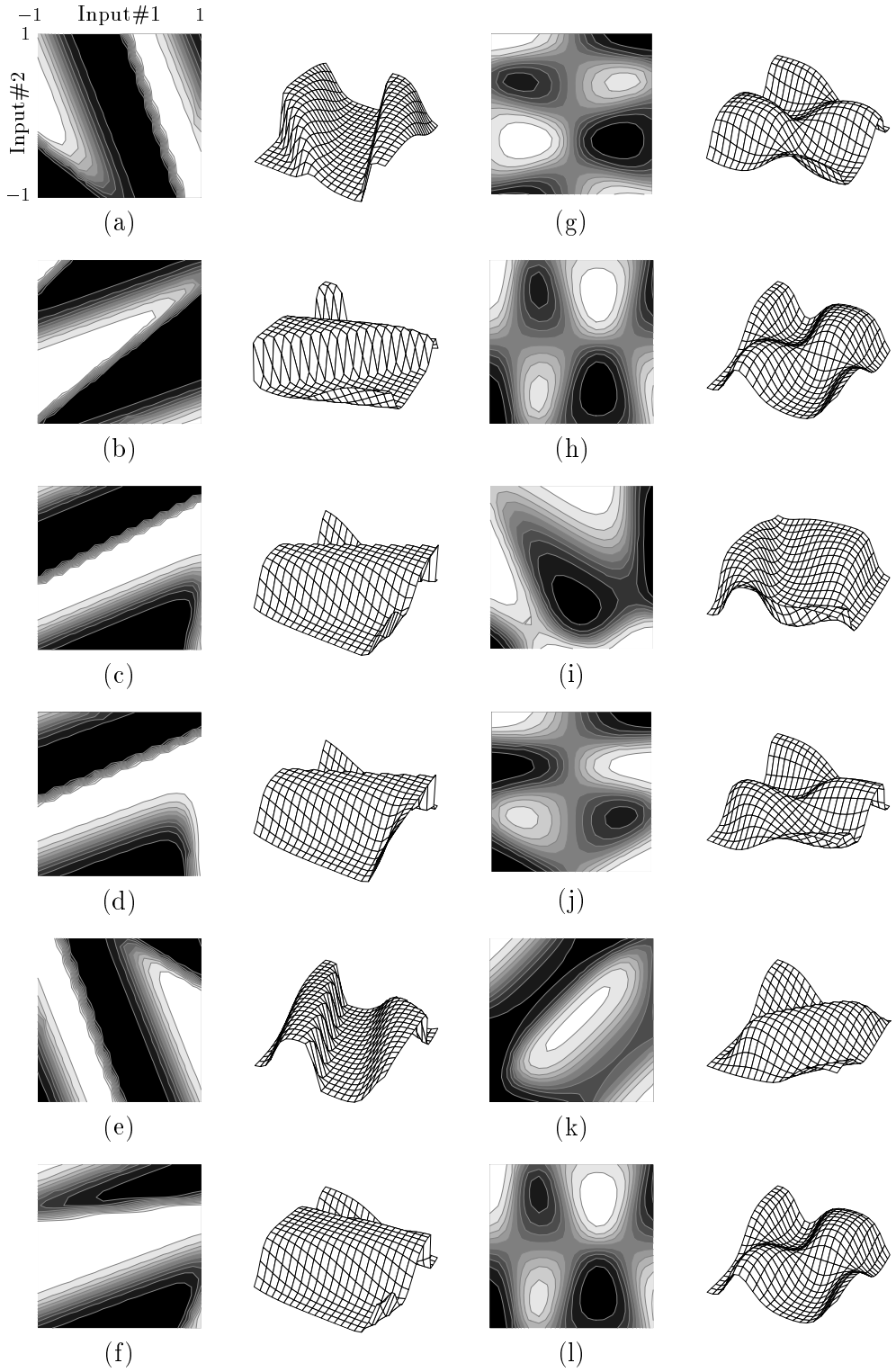


Figure C.13 Decision surfaces after 6 consecutive training runs on problem *E*: $\overrightarrow{2{:}2{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}4{:}1}$ network with integer weights (g-l).

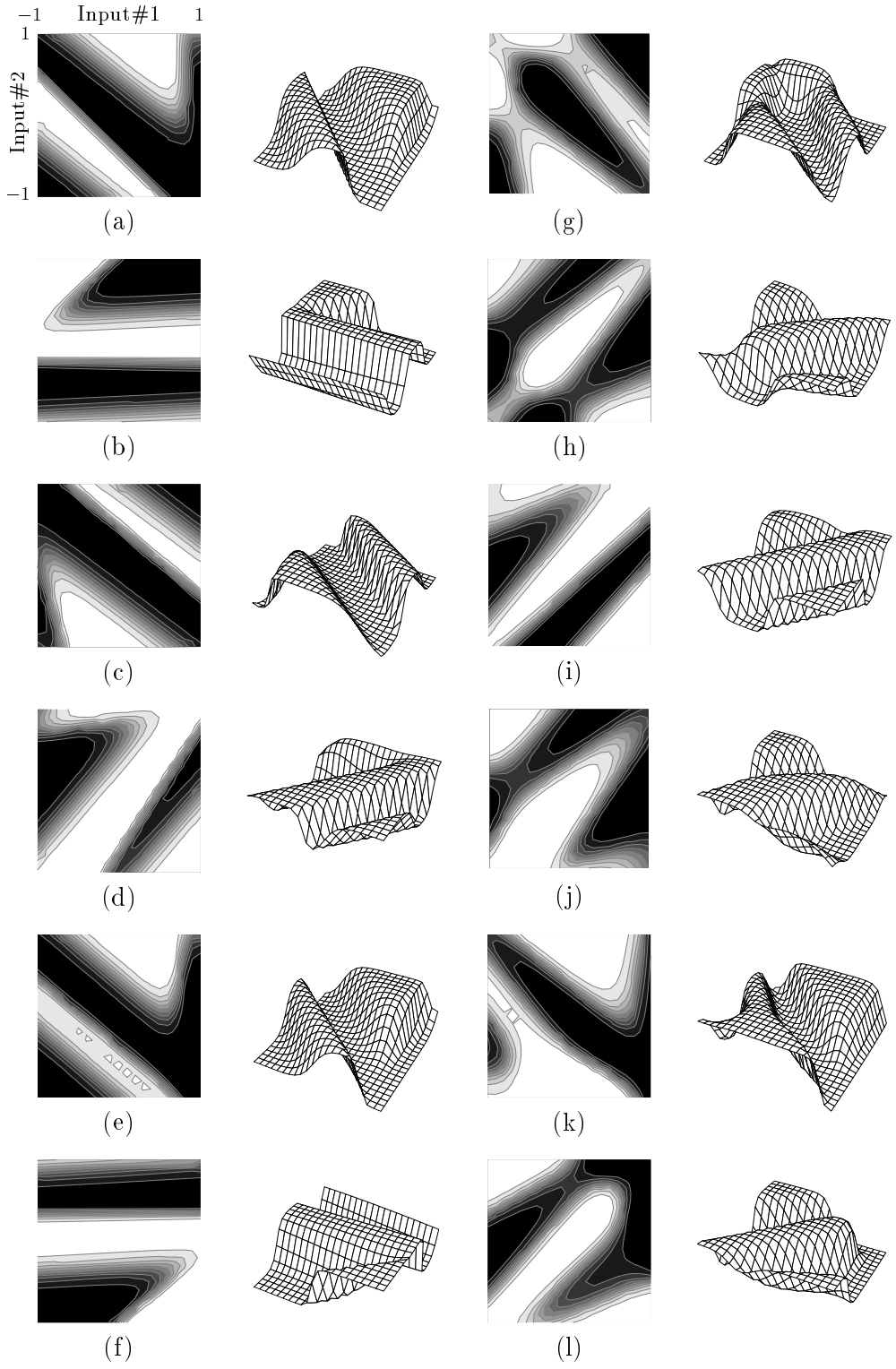


Figure C.14 Decision surfaces after 6 consecutive training runs on problem *E**: $\overrightarrow{2{:}2{:}1}$ network with double-precision weights (a-f); $\overrightarrow{2{:}5{:}1}$ network with integer weights (g-l).

D

Publications from This Thesis

This appendix includes the two papers which have been published as a result of the author's work on this thesis. The material presented in these papers is a condensed version of Chapters 3 and 4. The first paper also includes Chapter 6's IWN results on the MONK's benchmark.

A. H. Khan and E. L. Hines. Integer-weight neural nets. *Electronics Letters*, **30**(15): 1237–1238, July 1994.

A. H. Khan and R. G. Wilson. Integer-weight approximation of continuous-weight multilayer feedforward nets. *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 392–397, Washington, DC, June 1996. IEEE Press, New York, NY.

The following review also appeared during the author's work on this thesis.

A. H. Khan. Book Review: Fausett, L., Fundamentals of Neural Networks, (Prentice-Hall, 1994) in *The Computer Journal*, **37**(5):472, 1994.

Integer-Weight Neural Nets

Altaf H. Khan
Dept. of Engineering, Univ. of Warwick, Coventry, CV4 7AL, England
Dept. of Electrical Engineering, Univ. of Engineering & Technology, Lahore, Pakistan

Evor L. Hines
Dept. of Engineering
Univ. of Warwick, Coventry, CV4 7AL, England

ABSTRACT

Integer-weight neural nets (IWNN) are better suited for hardware implementation as compared with their real-weight analogues. We present a learning procedure for generating multilayer IWNNs having all weights in the set {-3, -2, -1, 0, 1, 2, 3}. The performance of this procedure was evaluated on XOR, encoder/decoder, and the MONK's benchmark. The IWNNs were found to be as capable as their real-weight counterparts regarding generalisation performance.

1. Introduction

Neural nets (NN) having integer weights and offsets are easier to implement in electronics as well as in optics. Integer weights in the range [-3, 3] can be represented by just 3 binary bits. This property reduces the amount of memory required for weight storage in electronic implementations. Moreover, in the special case of IWNNs with binary inputs, neurones in the first hidden layer have a very limited number of possible output states. Therefore, a neurone in that layer requires only an integer multiplier and adder, and a short look-up table for the transfer function, resulting in a simple electronic implementation. In optically implemented NNs, where weight values are represented as grey-scale masks or voltage levels for spatial light modulator, the complexity of implementation is again reduced because of integer weights. In this letter, a learning procedure for constructing multilayer IWNNs is discussed. Simulation results on the XOR, encoder/decoder, and MONK's benchmark are presented to illustrate its validity.

A multiple-thresholding method has been proposed for generating discrete-weight NNs [1], [2]. In this simple method, the continuous weights of a fully trained NN are quantised using a multiple-threshold non-linear function. This technique was used for training with ternary weights [1] and it was found that a large percentage of the resulting NNs failed to perform correctly when the weights were quantised. An alternative technique is a modification of the standard error Back-Propagation (BP) algorithm [3]. In this modified version, weight updates occur only if the required modification is large enough to move a weight from one discrete level to another [4], [5]. The Continues-Discrete Learning Method [5] follows a more fruitful strategy. In this method, a trained continuous-weight network is quantised, and then trained again. This cycle is repeated until the network converges. In addition, Marchesi et al. [6] have proposed a learning algorithm to generate NNs with powers-of-two or sum of powers-of-two weights and unrestricted offsets. This method simplifies the multiplication operation for neurones.

2. Integer-Weight Learning

The NN is initialised with small random weights and the learning process is started with little attention to weight quantisation. The quantisation mechanism slowly comes into play as the NN starts moving towards convergence, and becomes stronger as learning progresses. The on-line BP with momentum algorithm is used for the minimisation of the output error. The weight quantisation part of the training procedure consists of two mechanisms. The first mechanism determines the integer that a given weight is closest to. It then minimises the quantisation error of the weight with respect to that integer in the mean-squared sense. The change in weight is given by:

$$\Delta w = \chi(q - w),$$

where

q = integer closest to w, $q \in \{-3, -2, -1, 0, 1, 2, 3\}$,
χ = rate of weight quantisation.

The second mechanism acts as a black-hole centred at an integer. If a weight falls within the black-hole radius, it's value is forced to the centre-value of the black-hole. The weight quantisation rate and the black-hole radius are computed before each learning epoch, and are exponentially dependent upon the error of the output neurone with the maximum error (E_m)

3. Local Minima

The standard BP algorithm sometimes gets stuck in local minima [3]. The superposition of the weight quantisation process on BP results in even more local minima. At the start of integer-weight learning, E_m is large, and the output error minimisation process dominates. Conversely, the weight quantisation process has the upper hand when E_m is small. At intermediate values of E_m, the two processes may nullify each other's effect - resulting in a local minimum. To avoid local minima, the weight quantisation process is augmented by a perturbation process:

$$\Delta w = \chi(q - w)\tan(RND),$$

where

RND is a random number selected uniformly from $(0, \pi/2)$.

This perturbation strategy was very successful, except in cases where an IWNN was generated with an unacceptably large E_m. The solution we adapted in these cases was to strengthen the error minimisation process. This was accomplished by temporarily boosting the BP learning rate. A boost-up factor of 20 for one learning epoch worked quite well in our simulations.

4. Simulation Results

Fully connected feedforward NNs with hyperbolic tangent non-linearities in the hidden and output neurones were used for all simulations. The training data was scaled to the range [-1, 1]. Training was stopped when E_m was less than a prespecified value, ϵ, and all the weights had integer values. The weight quantisation rate and the black-hole radius of the form $\alpha e^{(\epsilon - E_m)\beta}$ was used. The values of α and β were empirically determined and were found to be problem insensitive.

Table 1 Weight quantisation parameters

Parameter	α	β
Weight quantisation rate, χ	0.001	16
Black-hole radius	0.1	6

Table 2 Learning epochs

Problem	Min.	Max.	Avg.	Median
XOR	15	69	31	27
Enc/Dec-4	10	42	22	21
Enc/Dec-8	129	1420	529	383

Table 3 Comparison of generalisation performance on the MONK's problem[a]

Problem	BP	BP with weight decay	Cascade correlation	Alopex	Integer-weight
#1	100%	100%	100%	100%	100%
#2	100%	100%	100%	100%	100%
#3	93.1%	97.2%	97.2%	100%	100%

[a] The BP, BP with weight decay, and cascade correlation results are from [7], and the Alopex results are from [8].

The functionality of integer-weight learning was tested by performing 25 simulations each on the 2-2-1 XOR, 4-2-4 encoder/decoder, and 8-3-8 encoder/decoder problems, with $\epsilon = 0.4$. The generalisation performance was evaluated on the MONK's benchmark which consists of three binary classification problems [7]. Problem #1 is in standard Disjunctive Normal Form (DNF), whereas problem #2 is similar to the parity problem. Problem #3 is also DNF but with 5% deliberate misclassifications in the training data set. IWNNs with various architectures were trained and best results were achieved with 17-4-1, 17-4-1, and 17-1 configurations, respectively. Perfect generalisation was attained in 171, 52, and 26 epochs, respectively. Due to the presence of misclassifications in the training data set, E_m was replaced by RMS error in the simulations for problem #3. A comparison of the generalisation performance of this integer-weight learning procedure and other real-weight learning algorithms on the MONK's problems is presented in Table 3. It is clear from Table 3 that integer-weight learning generates NNs which are at least as capable as the best generated by real-weight learning algorithms.

5. Conclusions

An effective procedure for constructing IWNNs has been presented in this letter. The NNs generated by this procedure may have additional performance related benefits besides the hardware implementation advantage. For example, this learning procedure produces IWNNs with some weights having a value of zero [1]. Moreover, the number of zero-weights is higher if larger-than-optimal NNs are used for training. The reduced

[1] The IWNNs trained on the MONK's problem #1, #2, and #3 had 61%, 51%, and 67% zero-value weights, respectively.

number of effective weights combined with the fact that the weights are restricted to seven values, limits the complexity of the network. This should result in reduced over-fitting and improved generalisation performance [8].

Acknowledgement: One of the authors (AHK) is supported by a scholarship from the Commonwealth Scholarship Commission in the UK. Initial part of this work was supported by a grant from the Directorate of Research, UET, Lahore, Pakistan.

References

[1] Chieueh, T. D., and Goodman, R. M.: 'Learning algorithms for neural networks with ternary weights,' presented at the *First Annual Meeting of INNS,* Boston, MA, September 1988, abstract: Neural Networks, 1988, **1**, p. 166

[2] Woodland, P. C.: 'Weight limiting, weight quantisation & generalisation in multi-layer perceptrons,' *Proc. IEE First Int. Conf. Artificial Neural Nets,* London, 1989, pp. 297-300

[3] Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: 'Learning internal representation by error backpropagation,' in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* **1**, Rumelhart, D. E., and McClelland, J. L. Eds. (Cambridge MA: MIT Press, 1986), pp. 318-362

[4] Von Lehmen, A., Paek, E. G., Liao, P. F., Marrakchi, A., and Patel, J. S.: 'Factors influencing learning by backpropagation,' in Proc. IJCN, July 1988, pp. I-335-I-341

[5] Fiesler, E., Choudry, A., and Caulfield, H. J.: 'A weight discretization paradigm for optical neural networks,' in *Proc. Int. Cong. Opt. Sci. & Engg.,* 1990, **SPIE-1281**, pp. 164-173

[6] Marchesi, M., Benvenuto, N., Orlandi, G., Piazza, F., and Uncini, A.: 'Design of multi-layer neural networks with power-of-two weights,' *IEEE ISCS,* New Orleans: 1-3 May 1990, **4**, pp. 2951-2954

[7] Thurn, S. B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S. E., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R. S., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Van de Welde, W., Wenzel, W., Wnek, J., and Zhang, J.: 'The MONK's problems: A performance comparison of different learning algorithms', Carnegie Mellon University, CMU-CS-91-197, December 1991. The training and test data sets for the MONK's problems are available at the `ftp` site `ics.uci.edu`

[8] Hush, D. R., and Horne, B. G.: 'Progress in supervised neural networks: What's new since Lippmann,' *IEEE Signal Processing Mag.,* 1993, **10**, pp. 8-39

[9] Unnikrishnan, K. P., and Venugopal, K. P.: 'Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks', *Neural Computation,* 1994, **6**, pp. 469-490

Integer-Weight Approximation of Continuous-Weight Multilayer Feedforward Nets

Altaf H. Khan
Department of Engineering
a.h.khan@ieee.org

Roland G. Wilson
Department of Computer Science
roland.wilson@dcs.warwick.ac.uk

University of Warwick, Coventry, CV4 7AL, England

ABSTRACT

Multilayer feedforward neural nets with integer weights can be used to approximate the response of their counterparts with continuous-weights. Integer weights, when restricted to a maximum magnitude of 3, require just 3 binary bits for storage, and therefore are very attractive for hardware implementation of neural nets. However, these integer-weight nets have a weaker learning capability and lack the affine group invariance of continuous-weight nets. These weaknesses, although compensatable by the addition of hidden neurons, can be used to one's benefit for closely matching the network complexity with that of the learning task. This paper discusses theses issues with the help of the decision and error surfaces of 2-D classification problems of various complexities, whose results suggest that in many cases, limited weight resolution can be offset by an increase in the size of the hidden layer in the network.

1. Integer-weight Nets

The response of a feedforward multilayer neural net having continuous weights (CWN) can be approximated with one having discrete weights in one of two ways [1] – by either allowing the number of discrete levels to grow [2, 4], or increasing the number of hidden neurons. This paper, while discussing both, will emphasise the latter of the two approaches, and will mainly be concerned with weights having small integer values. The results suggest that in many cases, finite weight resolution can be offset by an increased number of hidden neurons.

Integer weights in the range [-3, 3] require just 3 binary bits for storage. Moreover, if the inputs are restricted to the set {-1, 1}, the neurons in the first hidden layer require only sign adjustment for multiplication operations, and only integer addition. The transfer function of these neurons, e.g. tanh, can be very accurately implemented by using a lookup table with only 4 entries augmented by sign adjustments. The addition and multiplication operations in the output layer neurons of a single-hidden-layer net are not as simple as that for the hidden ones, but can be streamlined, keeping in mind the small number of possible output levels of the hidden neurons. For classification tasks, the transfer function of the output neurons can be implemented as a simple threshold. These features of the integer-weight net (IWN) make it very attractive for efficient hardware implementation. It should, however, be stressed here that these benefits are valid only when the IWN has been trained, as the learning task still requires full-precision arithmetic.

Discrete-weight nets are also attractive because the amount of information stored in their weights can be quantified [1]. The CWN can store an infinite variety of information, whereas the discrete-weight nets have only a limited capability. Varying

the discretisation scheme shows how much complexity is required to get a reasonable approximation in a given learning task. This can be a significant step towards understanding the fundamental relationship between approximation and memory.

The purpose of this paper is to explore the capabilities of discrete-weight nets on a set of classification problems, which are generalisations of the conventional XOR problem. It is shown that provided a suitable quantisation interval is chosen, a discrete-weight net can be found which performs as well as a CWN, but that it may require more hidden neurons than its continuous-weight counterpart.

The collection of learning heuristics used for the simulations in this paper is a modified version of the one described in [3]. The weight quantisation process is superimposed on the vanilla back-propagation with momentum error minimisation procedure. This weight quantisation process consists of two distinct mechanisms. The first mechanism determines the integer that a given weight is closest to. It then gradually reduces the difference of the weight with respect to that integer. The second mechanism acts as a black-hole centred at an integer value. If a weight falls within the black-hole radius, its value is forced to the centre-value of the black-hole. The weight quantisation rate and the black-hole radius are computed before each learning epoch, and are exponentially dependent upon the negative of the RMS error.

2. Approximating Continuous-Weight Perceptrons with IWNs

A perceptron with continuous weights has the ability of implementing an infinite variety of hyperplanes in its input space. On the other hand, a perceptron having W integer-weights in the range $[-K, K]$, is limited to $O\left(K^W\right)$ choices (see Figure 1). This restriction is the reason for the lack of the affine group invariance in integer-weight perceptrons. The addition of a hidden layer, however, is sufficient to restore this invariance. All ten data sets shown in Figure 2 are linearly separable and therefore can be classified correctly by a continuous-weight perceptron. IWNs with increasing number of hidden neurons were used to classify these data sets and it was found that the RMS value of the output error decreased logarithmically with the number of neurons (see Figure 3). It should be noted here that integer-weight nets with zero and one hidden neuron have different mapping abilities, which is contrary to the continuous-weight case. The IWN with a single hidden neuron has a richer mapping capability as compared with the one with no hidden neurons.

3. Approximating Continuous-Weight Multilayer Nets with IWNs

In the work reported here, mappings of the form $f : \mathcal{R}^2 \rightarrow \{-1, 1\}^1$, $\mathcal{R}$ being a closed interval $[-1, 1]$, were used for comparing the IWN and CWN decision surfaces for a set of 10 classification problems (Figure 4) which, were used for numerous training runs on 2:h:1 networks, with and without skip-layer connections.

Problems D, D^*, E, and E^* are of the same complexity, as far as the minimum number of required dichotomies is concerned. For all four problems, the CWN requires two hidden neurons with skip-layer connections and four without (Table 1). For the IWN, however, the numbers are a bit more variable, and larger. This is due to the inability of the IWN to implement dichotomies at arbitrary angles and for arbitrarily close training data points. It should, however, be pointed out that this inability can always be compensated for by the addition of hidden neurons. In a way, this drawback can be viewed as a beneficial feature of IWNs. This feature provides a finer control on

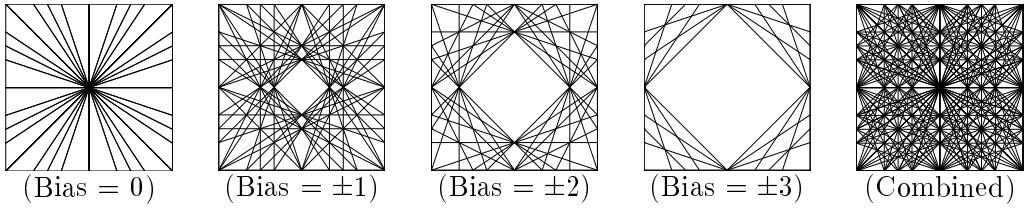


Figure 1 The set of decision boundaries of an integer [-3, 3] weight 2-input perceptron with bias. Some of the possible 7^3 decision boundaries lie outside the {(-1, 1), (1, 1)} square, and therefore are not shown.

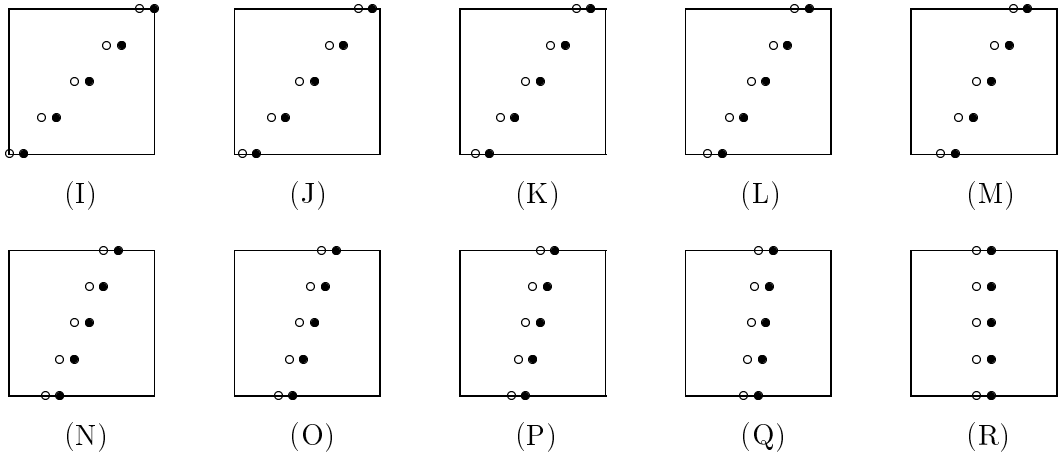


Figure 2 Linearly separable data sets with decision boundaries at gradually varying angles.

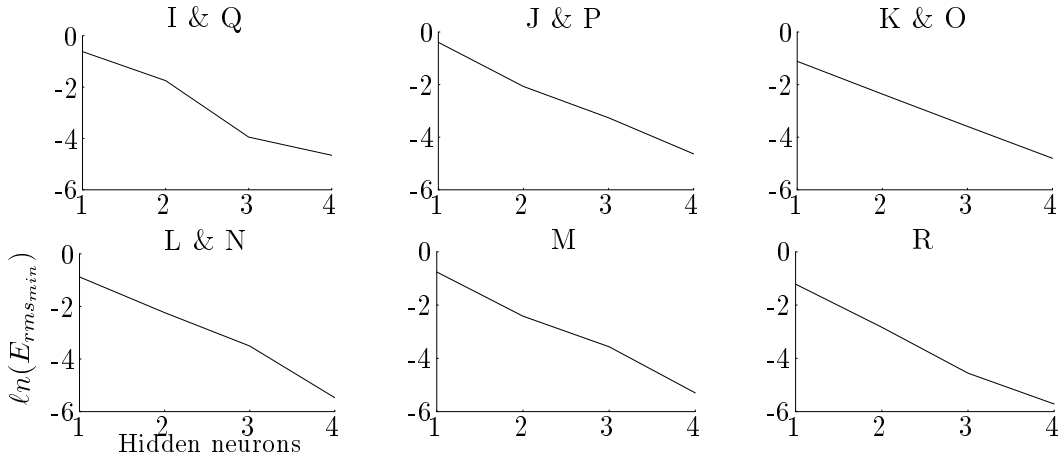


Figure 3 IWN minimum RMS error as a function of the number of hidden neurons for the data sets shown in Figure 2.

matching the network complexity with that of the learning task.

Problems D and E have the same number of dichotomies, but are different in terms of the angles of the dichotomies. E requires angles in a certain narrow range, whereas D can be implemented with a wider range of angles, and therefore is an easier learning task. A similar comparison can be drawn with respect to E and E*, with the added difficulty of smaller inter-point distances in E*.

A hexagon shaped training set was used to further study the effect of the angle of the training data dichotomies on the learning ability of the IWNs. The hexagon data set of Figure 5 was rotated around its centre in 5° steps, and $\overrightarrow{2\text{:h:}1}$ (i.e. 2:h:1 net with skip-layer connections) IWNs were trained on it. The results of those training runs are presented in Table 2. Only two data configurations, ones with rotations of 15° and 45°, were learnable with one hidden neuron. These data configurations resulted in exactly the same error surfaces except for a 90° rotation from the former to the latter. This, once again, shows that IWNs find it difficult to draw dichotomies at certain angles, favouring a discrete set of angles instead.

Almost all of the integer-weight training runs resulted in some repeated decision surfaces. For example, all skip-layer training runs on problem C* resulted in exactly the same decision surface. Similarly, all of the skip-layer runs for problem D resulted in similar decision surfaces, differing only by a rotation of a multiple of 90° (see Figure 6 g-l). The repetition of surfaces suggests that the complexity of the network is exactly matched with that of the target problem. Therefore, the problem solution, i.e. the decision surface, can be represented in only one way by the network. The reason for the repetition with extra rotation is certainly the 8-fold symmetry of D. Most of the problems resulted in some duplicated decision surfaces for IWNs, whereas none of the continuous-weight training runs resulted in any repetitions.

4. Error Surfaces

Another way of comparing discrete-weight nets and CWNs is via their error surfaces. Discrete-weight error surfaces are just the low-resolution sampled versions of the CWN error surfaces, as can be seen in Figure 9.

CWN error surfaces can be analysed to find out whether a network with a certain weight-resolution will be able to properly represent a given input/output mapping. If the CWN error surface contains sharp minima at points which are missed by the low-resolution sampling, then the discrete-weight net will not be able to represent the mapping with a reasonable accuracy. Modified versions of the XOR problem were used as test problems to study this phenomenon. These modified versions were produced by gradually bringing the four data points close to each other (see Figure 7). As the data points are brought closer, the slope of the error surface in the vicinity of the global error minimum becomes steeper. The minimum looks like the top of a funnel when the points are far apart, as in Figure 7 XOR1, and looks more like the middle of a funnel when the points are close by, as in XOR$\frac{1}{8}$. A $\overrightarrow{2\text{:}1\text{:}1}$ network with symmetric weights (Figure 8) was used to find the depth of the global minima for a number of weight resolutions. Symmetric weights were used to reduce the free parameters of the network from seven to five. It is quite clear from Table 3 that, in the case of XOR$\frac{1}{8}$, the error minimum has such a narrow collection region that the IWN can not see it because of low-resolution weight sampling, and a network with a weight resolution of at least 0.125 is required to correctly classify all the data points.

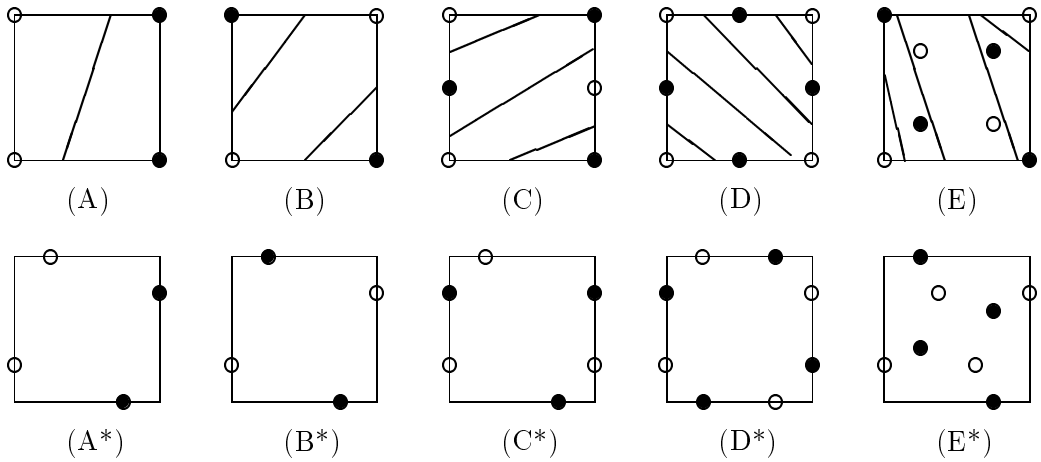


Figure 4 Training sets used for comparing the learning capabilities of IWNs and CWNs. The thick lines are the examples of the minimum number of possible dichotomies. *A*–E** are slightly deformed versions of *A–E* with smaller inter-point distances.

Table 1 Comparison of the minimum number of hidden neurons required by CWNs and IWNs for learning problems of increasing complexity. A training run was considered successful when the output value was within 20% of the {-1, 1} targets

		Minimum number of hidden neurons			
		with skip-layer connections		w/o skip-layer connections	
Problem	Dichotomies	$\overrightarrow{\text{CN}}$	$\overrightarrow{\text{IWN}}$	CN	IWN
A	1	0	0	-	-
A*		0	0	-	-
B	2	1	1	2	2
B*		1	1	2	2
C	3	1	1	3	3
C*		1	1	3	3
D	4	2	2	4	4
D*		2	3	4	4
E	4	2	4	4	4
E*		2	5	4	5

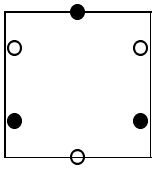


Figure 5 Hexagon training set with 0° rotation.

Table 2 Comparison of the minimum number of hidden neurons required by an IWN(with skip-layer connections) for learning the hexagon data set as it was rotated by 5° steps

Rotation	0°	5°	10°	15°	20°	25°	30°	35°	40°	45°	50°	55°	60°
Hidden neurons	2	2	2	1	2	2	2	2	2	1	2	2	2

5. Conclusions

Decision and error surfacers were used to explore the discrete-weight approximation of CWNs. Although most of the conclusions of this paper are based on integer-weight results, they are equally valid for other levels of discretisation.

The CWN can store an infinite quantity of information, whereas the discrete-weight net's resolution is finite. The latter does, however, has the advantage of efficient hardware implementation. Moreover, it provides an interesting way of quantifying information stored in a network [1]. Changing the discretisation scheme of the weights of a fixed-size network, or keeping a fixed discretisation scheme while changing the size of the network, are alternative techniques for determining the amount of network complexity required for approximating the response of a CWN to a specified tolerance. These two competing techniques can be used to understand the fundamental relationship between the approximation error and the storage capacity required to achieve that error. In addition, the level of control available on the complexity of discrete-weight nets can be exploited to one's benefit. The network designer can select a network with a complexity that matches more closely with the complexity of the learning task. This will result in improved generalisation performance.

Acknowledgement: Khan's work was supported by the Commonwealth Scholarship Commission in the UK and the University of Engineering and Technology, Lahore, Pakistan.

References

[1] R. W. Brause. The error-bound descriptional complexity of approximation networks. *Neural Networks*, 2(6):177–187, 1993.

[2] E. Fiesler, A. Choudry, and H. J. Caulfield. A weight discretization paradigm for optical neural networks. In *Proc. of the Int. Cong. on Opt. Sc. and Engg.*, pages 164–173, Bellingham, Washington, 1990. SPIE.

[3] A. H. Khan and E. L. Hines. Integer-weight neural nets. *Electron. Lett.*, 30(15): 1237–1238, July 1994.

[4] H. Yoo and R. L. Pimmel. Weight discretization in back-propagation neural network classifiers. In C. H. Dagli et al., eds., *Intel. Engg. Sys. through ANNs.*, pages 167–172. ASME Press, New York, 1991.

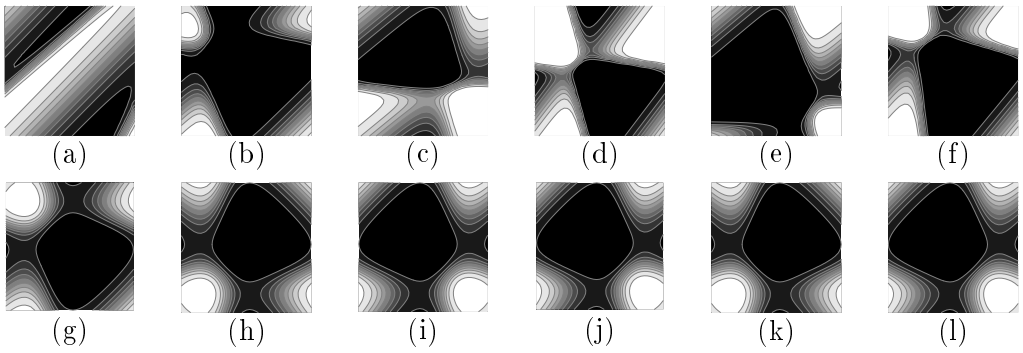


Figure 6 Decision surfaces after 6 consecutive training runs on problem D: $\overrightarrow{2{:}2{:}1}$ net with double-precision weights (a-f); $\overrightarrow{2{:}2{:}1}$ net with integer weights (g-l).

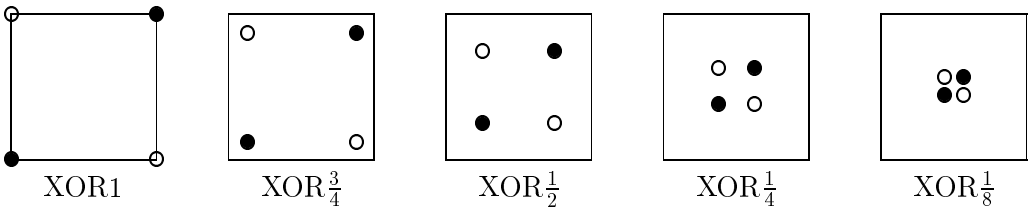


Figure 7 XORx training sets.

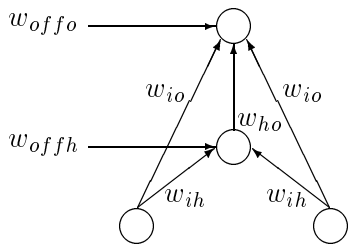


Figure 8 Symmetric weight $\overrightarrow{2{:}1{:}1}$ XOR network.

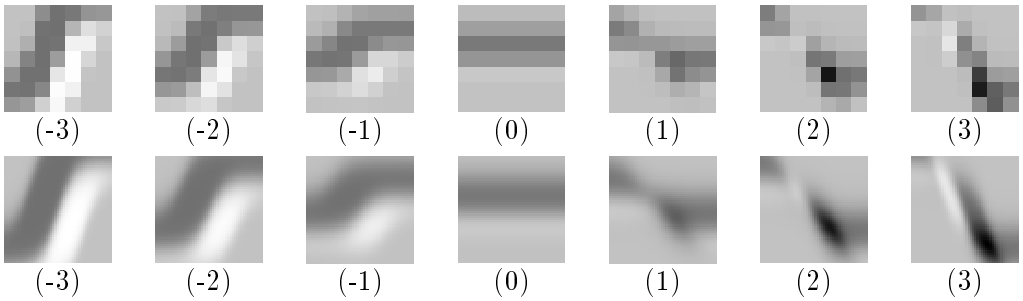


Figure 9 Error surfaces showing the global minimum for XOR$\frac{1}{8}$. The 7 images on the top are for integer weights and the rest are for a weight resolution of 0.125. $w_{ih} = 3$. $w_{io} = -3$. θ_h is plotted along the horizontal axis. θ_o is plotted along the vertical axis. w_{ho} is the figure in the brackets.

Table 3 Global error minima (each of multiplicity 4) as a function of weight resolution. The italicised figures indicate that one or more training vectors were *misclassified* at these global minima

	Weight resolution ($w_{min} = -3,\ w_{max} = 3$)								
	6	3	2	1.5	1	0.75	0.5	0.25	0.125
	Number of required binary bits								
	1	2	2	3	3	4	4	5	6
Problem	E_{rms}								
XOR1	*0.879*	*0.875*	*0.501*	0.097	0.210	0.097	0.097	0.097	0.097
XOR$\frac{3}{4}$	*0.722*	*0.722*	*0.719*	0.192	0.177	0.134	0.104	0.104	0.104
XOR$\frac{1}{2}$	*0.875*	*0.875*	0.212	0.148	0.212	0.148	0.148	0.139	0.139
XOR$\frac{1}{4}$	*1.167*	*0.946*	*0.736*	*0.711*	0.540	0.474	0.465	0.446	0.430
XOR$\frac{1}{8}$	*1.088*	*0.991*	*0.977*	*0.958*	*0.787*	*0.859*	*0.787*	*0.787*	0.784

Index