Butler, E. (2000). A dynamical study of the generalised delta rule. Unpublished PhD, U. of Nottingham.

A Dynamical Study of the Generalised Delta Rule

By Edward Butler, BA

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy, May, 2000

List of Contents

Abstract

The generalised delta rule is a powerful non-linear distributed learning procedure capable of learning arbitrary mappings for artificial neural networks of any topology. Yet, the learning procedure is poorly specified in that it cannot specifically guarantee a solution for all solvable problems. This study focuses on developing a benchmarking procedure for the generalised delta rule that provides a visualisation of the complete dynamics of the procedure and allows optimisation of all the variables that comprise the system functions together. A number of dynamical modes of convergence for the procedure are shown, in particular universal convergence to global error minima. In a number of experiments with small networks, the procedure was found to exhibit regions of universal global convergence for particular system parameters. With each problem examined, a particular value or range of values for the learning rate parameter was found that tunes the network for optimal learning success. In conclusion, it was found that small values of the learning rate parameter are not necessarily optimal for obtaining global convergence. It is further conjectured that feedforward generalised delta rule networks have enough representational capacity to map any combinatorial Boolean logic function, that a convergence proof should exist for these problems, and that the long term behaviour of the procedure should be tractable under universality theory.

Acknowledgements

Thanks to Han, Ian and Nigel for starting it all, and to everyone at Nottingham University for their expert help. I would especially like to thank my examiners, Mark Plumbley and Frank Ritter, for their extensive positive feedback, which was incorporated into this thesis.

List of Symbols

net_x	Total input to unit *x*.
T_x	Target value for unit *x*.
O_x	Activation value for unit *x*.
w_{xy}	Weight value from unit *x* to unit *y*.
θ_x	Bias term for unit *x*.
δ_x	Error term for unit *x*.
i.j,k	Unit indices.
e^x	Exponential function of *x*.
$Log_n x$	Natural logarithm of *x*.
λ	Learning rate parameter.
E, SSE	Sum squared error.
F	Feigenbaum number.
H	Entropy.

Chapter 1

Introduction

We are interested in the workings of the brain. The fact that the brain is a massively parallel neural system, and performs computation so effectively, suggests that it may be possible to attain similar capabilities in artificial systems based upon the design principles of neural systems. The complex and massively distributed structure of a natural neural system is difficult to conceptualise, but the obvious emergent intelligent characteristics of real neural systems suggests that artificial models of intelligence should be distributed information processing models. Such systems should benefit from the associated properties of fault tolerance, graceful degradation, learning, and the large potential for associative memory storage.

Hofstadter (1980) was one of the first to call for an examination of the sub-cognitive or “microstructural” cognitive level, stressing that it may be important to understand the microstructure to gain insight into the workings of the brain. This view of artificial intelligence implicitly assumes that the modelling approach should be bottom-up as opposed to top-down. For example, if we were to attempt to reverse-engineer some complicated, high-level software, we could first work out what the lowest-level instructions were, and incrementally build up our knowledge from there.

This bottom-up approach is used extensively in artificial neural network research. A criticism of the artificial neural network approach is that it concentrates on the small details, possibly missing the wider picture. This is a criticism that could be used against any bottom-up approach. A counter-criticism is simply that we must start somewhere.

One key to building up a judicious model, when our model is a natural system that is difficult to conceptualise, is surely to model only those features that can be justified in a computational or information-theoretic sense, rather than implemented for reasons of biological verisimilitude. The question is what are the necessary or essential features of true neural systems that make them such powerful information processors.

1.1 The Promise of the Neural Approach

A lot of artificial intelligence research is devoted to systems that learn simple input/output functions in unknown or complex environments. These systems would learn by trial and error, and generate a set of rules for optimal performance. The kind of rule-induction intelligence that these systems attempt to model is a basic paradigm for modelling intelligence with artificial neural networks.

A very interesting characteristic of artificial neural networks is that the learning component in such systems gives them a self-organising and self-programming dynamical behaviour. This ‘bootstrapping’ aspect of a self-organising artificial neural system is fascinating because it has similarities to the way an organism adapts to its environment.

The dynamics problem of sequence learning in artificial neural networks is akin to the problem of finding a sequence of operations like those of a computer program, to compute some general function, given only the input/output patterns. This is a very powerful idea, since it views the network as an 'automatic programmer' trying to find a sequence of instructions that will perform some specified task within an arbitrarily defined environment. The ‘Holy Grail’ of this research approach would be a ‘black box’ neural computer – such a device would require no programming knowledge at all, but would learn by ‘experience’.

1.2 Early Linear Learning Algorithms

Due to the extreme complexity of the biological neurone, gross simplifications have been made to allow an artificial neurone to be simulated in a realistic manner. McCulloch and Pitts (1943) described the first synthetic model of the biological neurone. In an attempt to capture the bivalent nature of the output of a biological neurone, they employed a bipolar activation function that operated on the sum of weighted binary inputs. As a result, the neuron acted like a simple switch that was either on (1) or off (0). Using the McCulloch-Pitts model, it was shown that simple networks could be constructed that perform the basic logic operations AND, OR, NOT, NOR, and NAND.

Later linear methods for modelling neural networks, such as the perceptron convergence procedure proposed by Rosenblatt (1961) and studied extensively by Minsky and Papert (1969, 1988) were limited in that the simulated neurone units could only use input and output layers. Extra hidden layers of units were possible, but these units had to have the values on their connections set by hand. No learning could take place on the connections that were fixed, because the linear procedure could not determine the correct adjustments to make to the weights before those units – and this turned out to be a limit for early research.

Although the perceptron convergence procedure is limited to a single-layer of linear threshold units, it is capable of solving any problem or function that is linearly separable. Minsky and Papert (1969) showed that the perceptron convergence procedure guarantees that a set of weight values to solve any linearly separable function can be found, as long as that set of weights can in principle represent the problem.

Another early single-layer learning algorithm is known as the least mean square procedure (Widrow and Hoff, 1960). This learning procedure is not applicable to threshold units. Instead, the transfer function of each unit must be linear. A crucial part of the method is the computation of an error function, which is a measure of the sum squared error of all desired unit states minus the actual output states. The least mean square procedure learns input/output functions by changing the values of the weights by an amount proportional to the derivative of this error, with respect to each weight. This implements steepest gradient descent of error in the weight-space of the system.

Chapter 2

The Generalised Delta Rule

Non-linear distributed learning algorithms are an advancement upon earlier linear methods for artificial neural computation, such as the least-mean-square method, McCulloch-Pitts model and perceptron convergence procedure, but the non-linear methods use the same basic principles. The research presented here concentrates on the generalised delta rule learning algorithm, often referred to as back-propagation, a method independently derived by Werbos (1974), Parker (1985), Le Cun (1985) and Rumelhart, Hinton and Williams (1986a,b).

The generalised delta rule has become the network training method of choice for many cognitive modelling projects. Many artificial neural networks use some form of the generalised delta rule algorithm (for example, see Melnik and Pollack, 1998; Chang and Mak, 1999). The procedure is probably the widest-used learning rule for a number of reasons: it retains the clear paradigmatic simplicity of the perceptron convergence procedure; it makes it easy to find the appropriate connection pattern for artificial networks of any topology; and it is capable of learning arbitrary mappings.

Rumelhart, Hinton and Williams (1986a,b) developed and popularised the generalised delta rule learning procedure for networks of units with non-linear transfer functions. As with linear methods the procedure repeatedly adjusts the weights of the connections in the network to minimise the error between the actual and desired output states of the network.

The generalised delta rule is based in part upon the earlier linear method of the perceptron convergence procedure (see Rumelhart, Hinton and Williams, 1986a) but is distinguished from the perceptron convergence procedure by being able to more fully utilise hidden units between input and output units. A powerful feature of the generalised delta rule is that the hidden units have actual and desired states that are not directly specified by the task. The same weight-adjustment procedures can be applied to the hidden units, creating useful new features for difficult non-linear task domains.

The procedure is also distinguishable from earlier methods such as the perceptron convergence procedure by the ability to converge to error minima in networks of arbitrary topology, and there are generalisations of the procedure to cover many topologies such as the recurrent topology (for example see Giles, Miller, Chen, Chen, Sun and Lee, 1992; Tsung and Cottrell, 1993; Williams and Zipser, 1994; Chang and Mak, 1999; and Mak, Ku and Lu, 1999) and the cascade topology (Fahlman and Lebiere, 1989). The generalised delta rule is also "faster than earlier learning methods, but is still much slower than we would like" (Fahlman, 1988, p. 1). It is still, however, well known for being very slow (Qian, 1999).

The procedure progressively adjusts the weights of the connections in a network according to errors between actual and desired output states, and in the limit minimises such errors. This progressive reduction of input/output errors by the adjustment of the relative strengths of connections between processing units provides a simple and very effective model of distributed learning. The iterative reduction of error also gives the system the property of settling into, rather than calculating, a solution.

The procedure involves only computations local to each unit and is thus a true parallel method. The repetitive application of the procedure is guaranteed to minimise the squares of the differences between actual and desired output values, summed over the output units and all pairs of input/output vectors, subject to certain conditions. An example of a condition where the procedure is not guaranteed is where "the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimum" (Rumelhart, Hinton and Williams, 1986b, p. 535).

2.1 The Standard Generalised Delta Rule

Throughout this thesis, I will follow the nomenclature of Rumelhart, Hinton and Williams (1986a). The standard learning procedure, as defined by Rumelhart, Hinton and Williams (1986a), involves presenting a set of input patterns to the input units of the network. An input pattern or vector is presented to a network by setting the states of the input units. The states of the units in subsequent layers are determined by computing the net input to a given unit, and then applying an activation function to the computed net input to that unit.

Consider a unit indexed by j. Units in the previous layer will be indexed by i, and those in the next layer by k. First, the total input net_j to the unit is computed. This is a linear function of all the activation values, O_i, of units that are connected to j, multiplied by the weight values, w_{ij}, on those connections. If the unit is biased then we also include the bias term, θ_j, which is similar to a threshold. It is taken as the weight from a unit that is always on, in the sense that it has an effect on the total net input to the next unit. We imagine that the input along this weight is always set to one or minus one, or any arbitrary nonzero value.

$$net_j = \sum_i O_i w_{ij} + \theta_j \qquad (1)$$

After the calculation of the net input, we can apply the activation function. The activation function for each unit gives the unit a real-valued output, O_j, which is a non-linear function of (1). The most popular is a sigmoid function:

$$O_j = \frac{1}{1 + e^{-net_j}} \qquad (2)$$

Eq. (2) is sometimes called the squashing function. The use of a linear function (1) to summate the net input for a unit before applying the squashing function (2) greatly simplifies both the derivation of and the actual operation of the learning procedure.

Rumelhart, Hinton and Williams (1986b) assert that we do not have to use exactly the same equation as (2), “any input-output function that has a bounded derivative will do” (Rumelhart, Hinton and Williams, 1986b, p. 534). Sometimes other researchers (for example see Manausa and Lacher, 1991, 1993) use an arctangent function instead of the sigmoid.

Starting with the next layer on from the input layer, each unit computes a value for its output according to equations (1) and (2), then the activation functions of the next layer of units are computed, and so on, until all the activation functions in the output layer have been computed. This constitutes the forward pass, known as forward propagation.

When forward propagation is complete, the output units take on certain values, which are then compared to target values. This comparison, if unequal, generates an error term, which is then 'backpropagated' through the network, by altering the strengths of the connections between the units.

The error term δ for an output unit is the difference between the actual and desired output values, multiplied by the derivative of the squashing function. The error term for a hidden unit (for which there can be no directly specified target) is determined by the sum of the error terms of the later units to which it directly connects and the weights of those connections, also multiplied by the derivative of the squashing function.

Using the standard generalised delta rule, the derivative of the sigmoid squashing function (2) turns out to be the logistic function, O_j *(1 -* O_j*).* Therefore, the error term, δ_j, for each output unit, O_j, can be computed by multiplying this derivative with the comparison to the target value, T_j, for that unit:

$$\delta_j = O_j(1-O_j)(T_j - O_j) \tag{3}$$

The error terms for all subsequent layers are then computed, with the order of processing being the reverse of the forward pass. This second pass is called the backpropagation phase. The error signal δ_j for a hidden unit O_j is:

$$\delta_j = O_j(1-O_j)\sum_k \delta_k w_{kj} \tag{4}$$

Once all of the error terms have been calculated, the actual weight values for all of the connections used throughout the network can be updated:

$$\Delta w_{kj} = \lambda O_j \delta_k \tag{5}$$

The λ parameter is the learning rate or constant of proportionality. True gradient descent requires both that λ be infinitesimally small, and that the weights should be updated by the accumulated error over all input/output presentations. Breaking either of these conditions means that we depart to some degree from true steepest gradient descent. This departure from true may not be a problem, but it does suggest that for accurate steepest gradient descent we should use a very low learning rate and accumulate the error terms from all input/output presentations. Conversely, using a high learning rate or a different weight update regime may produce unexpected results.

Other methods have also been proposed for improving the speed of convergence of gradient descent learning algorithms. For example, the conjugate gradient method has been shown to be superior to the steepest descent method in a number of applications (Press, Teukolsky, Vetterling and Flannery, 1992). However, the conjugate method requires more storage of intermediate results than the standard generalised delta rule method, and is not a truly parallel method in the sense that the information needed to update a weight is not all contained in the pre- and post-synaptic units of the weight (Qian, 1999). This makes the algorithm less biologically plausible and harder to implement in hardware.

2.2 Topologies

Networks that use the procedure can be classified into two topological types. The simplest type is known as a layered feedforward network. In this type of network, we have a bottom layer of input units and a top layer of output units, with one or more layers of 'hidden' units in between. This type of network topology is constrained by not allowing a unit to have any input connections from the outputs of any unit in a higher layer. Thus, this topology avoids any closed (feedback) loops. A network that does not satisfy this constraint has a recurrent topology.

The generalised delta rule is performed using two distinct computational phases, a forward-propagation phase, and a back-propagation phase. In feedforward networks, the forward-propagation phase has roughly the same computational complexity as the back-propagation phase. In the recurrent topology, the back-propagation phase is usually much more complex. Generally, the two topology types are used for different purposes, the feedforward to learn arbitrary mappings from inputs to outputs and the recurrent to learn to produce input sequence/output sequence pairings. In this study, I shall consider only the simplest of feedforward topologies. Recurrent topology studies can be found elsewhere (see, for example, Giles, Miller, Chen, Chen, Sun and Lee, 1992; Tsung and Cottrell, 1993; Hochreiter and Schmidhuber, 1997; Melnik and Pollack, 1998).

Figure 1 shows the two simplest possible topologies for studying the generalised delta rule – networks that consist of just one input unit, one output unit, and one hidden unit (for short, a 1:1:1 network). Threshold or bias terms can be added to hidden units or output units by connecting the unit via a weight to a fixed non-zero value, usually one. Two possible 1:1:1 networks, with and without thresholds, are shown in Figure 1. The θ symbols in the lower network each represent an individual weight to a fixed bias value for that unit.

Figure 2 shows a 2:1:1 feedforward network topology. The network has two input units, one biased hidden unit, and one biased output unit. The network is fully interconnected, and has seven weights in total, including biases. This is a classic network for efficiently solving the exclusive-or problem, and a candidate for solving any 2-input Boolean function. Indeed, the 2:1:1 is the simplest network that can represent the XOR function (Sprinkhuizen-Kuyper and Boers, 1996a).

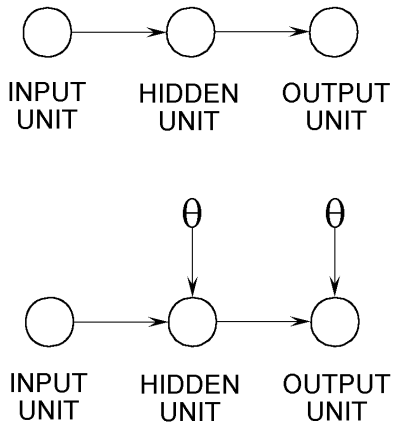


Figure 1. Two 1:1:1 network topologies. The first makes use of one input unit, one hidden unit, and one output unit, with two weights connecting them. The second network has extra weights (bias terms) on the hidden unit and output unit, making a total of four weights.

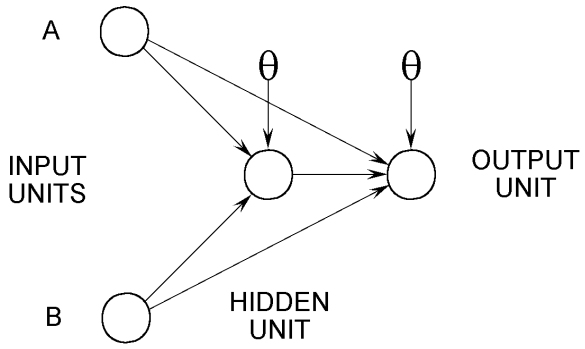


Figure 2. A 2:1:1 feedforward network. The network has two input units, one biased hidden unit, and one biased output unit. This network is fully interconnected, and has seven weights in total, including biases.

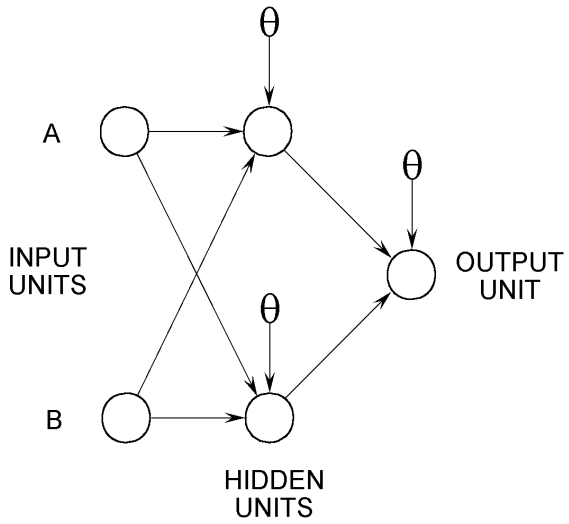


Fig 3. A 2:2:1 limited feedforward network. The network has two inputs, one biased output, and two biased hidden units, a total of nine independent weights including biases. There are no direct connections between the input and output units.

Figure 3 shows a 2:2:1 limited feedforward network topology. Since there are no direct connections between input and output, this network is at a disadvantage when compared to the fully interconnected version, in the sense that the output unit in Figure 3 has access only to the values of the hidden units and no direct connections to the input vector. The 2:2:1 limited feedforward network can be used as an example of one of the most popular benchmarks in the field of neural networks (Howes and Crook, 1999).

Sometimes, investigators compare the 2:1:1 fully interconnected topology with the 2:2:1 limited feedforward topology on a problem such as exclusive-or in order to more fully understand how these topologies can affect learning (see for example Sprinkhuizen-Kuyper and Boers, 1996a,b).

2.3 The Task Domain: Boolean Algebra

Boolean algebra is classical logic defined using algebraic terminology and operations, as proposed by George Boole in 1854. Claude Shannon was able to show how a standardised two-valued Boolean algebra could be used in the analysis and synthesis of digital computer circuits. He developed these ideas in a thesis, which was later published in the American Institute of Electrical Engineers Transactions in 1938. Standardised two-valued Boolean algebra can be used to represent the properties of digital circuitry, and is used extensively as a tool for digital computer design.

Standardised Boolean algebra requires a minimal set of postulates, and is defined over the set (TRUE, FALSE) and over the elementary Boolean operators AND (.), OR (+), and NOT (~), such that all operators are closed - all values must be either TRUE or FALSE. In digital computer design, we usually assign zero to FALSE and one to TRUE.

The truth table is a standard way of expressing logical relationships. For example, the AND operator or its truth table are ways of expressing the same relationship between two Boolean variables. If we group together all possible unary logic functions of a single Boolean variable $x = f(A)$ we end up with the truth table of the four primitive Boolean unary functions, as shown in Table 1. For example, the value of $x=f0\ (A)$ is always zero and hence produces the unary Boolean FALSE function.

Table 1. Unary Boolean Operators for x= f(A).

A	0	1	
f0	0	0	FALSE
f1	0	1	IDENTITY
f2	1	0	NOT
f3	1	1	TRUE

We can produce a similar truth table for all the binary Boolean functions, also called operators. This is show in Table 2. All binary Boolean operators have the property of associativity, where all operators of the same type are independent of evaluation order, and the property of commutativity, where the order of the operands is unimportant.

Table 2. Binary Boolean Operators for x=f(A,B).

A	0	0	1	1	
B	0	1	0	1	
f0	0	0	0	0	FALSE
f1	0	0	0	1	A AND B
f2	0	0	1	0	A AND ~B
f3	0	0	1	1	A
f4	0	1	0	0	~A AND B
f5	0	1	0	1	B
f6	0	1	1	0	A XOR B
f7	0	1	1	1	A OR B
f8	1	0	0	0	A NOR B
f9	1	0	0	1	A XNOR B
f10	1	0	1	0	~B
f11	1	0	1	1	A OR ~B
f12	1	1	0	0	~A
f13	1	1	0	1	~A OR B
f14	1	1	1	0	A NAND B
f15	1	1	1	1	TRUE

Any Boolean algebraic expression that is a well-formed formula has a complement that is also well formed. Expressions are comprised of terms, which are comprised of literals and operators. A literal is a complemented (*~x*) or an uncomplemented (*x*) variable or operand. A term is a sub-expression, often enclosed in parenthesis. The equation:

$$A = (x + \sim y) \cdot (\sim x + z) \tag{6}$$

Has four literals and two terms. The above equation is said to be in product-of-sums form, because it is comprised of two summed (ORed) terms that are ANDed together. The dot symbol used to denote AND is the same symbol used to denote multiplication, hence the name ‘product’.

Both sum-of-products and product-of-sums forms are called standard forms, and one can convert between the representations freely. Any Boolean equation can be written in either form. For example, converting the above product-of-sums equation into sum-of-products form gives:

$$A = \sim(\sim x \,.\, y) + \sim(x \,.\, \sim z) \qquad (7)$$

Both these equations are specifications of the exclusive-or Boolean function, which is function number six in Table 2. The exclusive-or and the equivalence (exclusive-nor) functions are just special combinations of AND, OR and NOT. In the design of digital systems the reliance of the use of the NAND or the NOR gate as a basic unit to implement an entire system stems from the sum-of-products/product-of-sums duality in Boolean algebraic representation.

A very important point about representing either the exclusive-or or exclusive-nor operators using Boolean algebra is that we always need at least two product terms in order to specify either function, whether we use either a product-of-sums or the equivalent sum-of-products form.

Minsky and Papert (1969) attempted to explain the computational capacities and limitations of simple networks of threshold logic units, binary input vectors and numerical weights. One of the ways of doing this is by defining the order of a predicate. The order is defined as the largest conjunction in the minimal sum-of-products form of the predicate (Minsky and Papert, 1969). Thus the exclusive-or operator is a predicate of order 2, whereas simple AND or OR operators are linearly separable and of order 1. As to the generalisation of the order of a predicate for the exclusive-or operator for any number of inputs (parity), we find that a sum of products form that represents parity of *n* inputs has at least one term of size *n*.

The standard Boolean algebra that we have been considering here is known as combinatorial, as opposed to sequential, logic. Combinatorial logic is a network whose output is solely dependent upon its inputs, with no feedback loops or memory elements. One can see immediately that there is an equivalence between the topological constraints of combinatorial logic and the constraints imposed by layered feedforward networks.

There is also a logical or computational equivalence between combinatorial logic and layered feedforward networks. For example, if we take the function of a single unit as a linear threshold device, we can see that the output of the unit will be zero (FALSE) until the net input to that unit is above a certain threshold. If the threshold is set so that the output becomes TRUE only if all the inputs to that unit are also TRUE, then the unit is essentially performing the AND function. We can similarly achieve the OR function, and if we allow units with positive or negative thresholds we can produce the NOT function. Both the NOR and the NAND functions can be therefore be effectively performed by a linear or semilinear threshold units.

The standard generalised delta rule was developed “with digital, two-valued learning in mind” (Rumelhart, Hinton and Williams, 1986a). When using the standard sigmoid activation function, the error derivative, $O(1 - O)$, reaches its maximum (1/4) for $O = 1/2$ and tends towards zero as O approaches zero or one. It follows that the maximum weight changes occur to weights fed into a unit that has an output value near 1/2, and that weight changes will tend to zero when an output value is near zero or one. In the long term, the units that comprise the system will therefore tend to take on values towards the extremes of zero or one, if required, making generalised delta rule systems that use the sigmoid activation function suitable for representing and learning digital logic functions.

A feedforward network of biased units using the standard generalised delta rule procedure with a sigmoid activation function should therefore have all the topological and computational resources available to be able to learn to compute any combinatorial logic function, given enough units to represent the product terms which would be required if the problem were represented using Boolean algebra. The relationship between linear threshold units and combinatorial logic was extensively studied by McCulloch and Pitts (1943).

We can minimise the number of product terms in a sum-of-products representation for any Boolean equation. To represent all the unary Boolean equations, the requirement is one term. For all the binary Boolean equations, the requirement is two terms, such as the case with, for example, exclusive-or. Therefore, it follows that the generalised delta rule procedure should in principle be able to learn any unary Boolean function using a 1:1:1 biased network (Figure 1), and any binary Boolean function using a 2:1:1 biased network (Figure 2). Such networks have enough representational capacity to represent these functions. More complicated functions will require more units and connections.

An important point to note here is that when learning digital functions, where we have target values of precisely 1.0 or 0.0 for true or false, we find that to achieve these targets the net input to a unit must eventually reach a value of plus or minus infinity. Therefore, most researchers use target values that are “backed off” from the extremes of 1.0 or 0.0. Typically, target values of 0.9 and 0.1 for true and false are used, leading to finite weight solutions.

Throughout this thesis I will study mostly digital problems involving target values of 1.0 and 0.0 for representing true and false. Although this is an uncommon testing condition, it is not unheard of, for example Sprinkhuizen-Kuyper and Boers (1996b) have provided a mathematical analysis of a network learning the exclusive-or function with infinite weight solutions. For the sake of completeness I also include some finite-weight solution studies in this thesis, using “backed-off” target values.

2.4 Theoretical Problems

In the conclusion to their proposal of the generalised delta rule, Rumelhart, Hinton and Williams (1986a) affirmed that they believed their algorithm was an answer to the challenge for multi-layered artificial neural networks put forward by Minsky and Papert (1969), that "Perhaps some powerful convergence theorem will be discovered" (Minsky and Papert, 1969). However, Rumelhart, Hinton and Williams (1986a) admitted that their procedure could not guarantee a solution for all solvable problems. This meant that the generalised delta rule was not, in the strictest sense, a direct generalisation of the perceptron convergence procedure, because the perceptron convergence procedure specifically *guarantees* a solution, if that set of weights exists.

In a later reprint of *Perceptrons*, Minsky and Papert (1988) have argued that there exists no proof of convergence for the generalised delta rule procedure. Moreover, they clearly state that they still believe that "the generalisation is sterile" (p. 231), i.e., they do not see any valid improvement over the original perceptron convergence procedure merely by employing a squashing function to make the activation function of a unit non-linear.

The main theoretical aims of this study are to show how these theoretical problems may be overcome. It is, for example, common knowledge that it is non-linearity that enables the generalised delta rule to perform non-trivial computations in the first place. Perhaps we may be able to show that Minsky and Papert (1988) were misplaced in their criticism of the non-linear generalisation.

Chapter 3

Empirical Problems

Problems associated with the generalised delta rule procedure are unfortunately many-fold. One of the known problems is that the procedure requires an unspecified and possibly very large number of iterations before the system settles into a solution state (Pollack, 1988; Qian, 1998; Chang and Mak, 1998). Another problem is that the optimal value for the learning rate parameter is unknown (Riedmiller and Braun, 1993). Still another is that the effect of the initial conditions and training regime upon the dynamics is not fully known. Moreover, when the system has converged to a solution state, the solution is not guaranteed to be globally optimal, and may have settled into a local minimum of error that is unusable (Pinkas and Dechter 1995).

A goal of this study is therefore to develop a method to overcome these problems and limitations. In the following sections I will attempt to examine each of these problems in order to clarify the situation, and see what factors need to be taken into account to solve the problems. For example, it is possible that we could re-specify the generalised delta rule method and develop a sound proof of convergence, taking all of these factors into account. It is also in principle possible that an acceptable convergence proof could be found without having to alter the definition of the generalised delta rule at all.

3.1 Selection of Training Regime

The choice for selection of training regime is basically between that of updating weights after every input/output training exemplar presentation, or updating the weights with a total of the accumulated increments after the entire set of exemplars has been presented. These regimes are also referred to as on-line training (Biehl and Schwarze, 1995) and batch training (Chang and Mak, 1999; Ampazis, Perantonis and Taylor, 1999).

It is possible that systems using the method of on-line training, that is, updating the weights after each individual training pattern, may give results sensitive to the presentation order. By definition, batch training, the method of updating after each epoch, must be independent of such effects. The method of updating after a whole epoch is also more conformal to true gradient descent, because the summed gradient information for the whole pattern set contains more reliable information regarding the shape of the entire error-function (Riedmiller and Braun, 1993).

3.2 Specification of Initial Conditions

In the original specification of the generalised delta rule, Rumelhart, Hinton and Williams (1986a,b) reason that we must start a network off with small random weights in order to "break the symmetry". According to them, initial weights of exactly the same value cannot be used, since symmetries in the environment are not sufficient to break symmetries in initial weights.

It can be argued that a very high initial weight range would lead to very long convergence times, since the derivative of the squashing function approaches zero for larger weights. In practice, it is therefore usual to set the initial weight values to random numbers with a uniform distribution between a small range of values. A small range is taken to be a nominal range of plus and minus x, x usually being around 0.5 (for example see Riedmiller and Braun, 1993; Ampazis, Perantonis and Taylor, 1999).

My own view is that this symmetry-breaking reasoning could be mistaken. For example, let us consider just how truly "random" any random initial conditions can be. For example, we know that all systems modelled on a digital computer must be initialised with weight values that are derived from a pseudo-random number generator, and so can never be "truly" random, because random number programs generate complex repeating sequences of values that just *appear* random to a greater or lesser extent (for example, see Marsaglia, 1993).

It follows that a system might be initialised with a configuration of pseudo-random numbers that, by accident, happens to be the configuration that solves the specific problem that the system is supposed to solve. A certain configuration of pseudo-random numbers from a given range may lead to a solution that is poorer, by some measure, than another set of numbers from the same range. Therefore, we are relying, to some extent, on the initial pseudo-random configuration "seed" being the one that somehow leads to a correct solution. The specification of the initial conditions for the generalised delta rule can in this light be seen as inadequate.

Kolen and Pollack (1990) studied the effects of varying the initial weight selection on simple feedforward networks using back-propagation, experimenting with a 2:2:1 network trained on the exclusive-or function. Using a measure they called *t-convergence* to refer to whether or not a network, starting at a precise initial configuration, could learn a Boolean function (correct outputs above or below 0.5) within *t* epochs, Kolen and Pollack varied the learning rate and the initial weight range.

Kolen and Pollack demonstrated that the magnitude of the initial condition vector is a significant parameter in convergence time variability. They further suggested extreme sensitivity of the generalised delta rule to the initial weight configuration. Kolen and Pollack were led to the conclusion that there were fractal-like boundaries between convergence and non-convergence, due to the existence of multiple attractors (solutions), the value of the learning parameter, and the non-linear determinism of the gradient descent approach. They went further to suggest that rather than a gradient-descent metaphor with local minima to get stuck in, we should instead think of a many-body metaphor, with many-bodied interactions implying that the system trajectory through weight-space will not be a simple one.

Kolen and Pollack implore the cognitive science community, when making claims about backpropagation, that “the initial conditions for the network need to be precisely specified or made publicly available”. Their most interesting point was that “it may also turn out that search methods which harness ‘strange attractors’ ergodically guaranteed to come arbitrarily close to some subset of solutions might work better than methods based on strict gradient descent.” (Kolen and Pollack, 1990, p. 272). I will take up this fascinating issue later in this study.

3.3 Selection of Learning Rate Parameter

The selection of the learning rate parameter when used with the standard generalised delta rule has been described as a ‘black art’ (Fahlman, 1988). Indeed, a short review of the literature surrounding the problem shows that this selection is generally regarded as an art rather than a science (for example see Fahlman, 1988; Turega, 1992; Manausa and Lacher, 1990, 1993; Riedmiller and Braun, 1993; Ampazis, Perantonis and Taylor, 1999; Chang and Mak, 1999). The general consensus seems to be that optimal parameter values are problem-specific, and should be as large as possible, but low enough so as to avoid any instability, wandering or oscillatory behaviour.

Manausa and Lacher (1990, 1993) conducted a number of studies to investigate learning rate and gain parameter sensitivity for back-propagation. The systems that Manausa and Lacher studied used an arctangent activation function, which is a standard sigmoid shape but also has a gain parameter, sometimes called a temperature parameter. They found extreme sensitivity of training time, and training success, to these parameters.

Manausa and Lacher (1990, 1993) make an interesting point about the size of the lambda parameter. A small step size will possibly minimise oscillation, and thus enhance convergence. On the other hand, increasing step size should enhance convergence in cases where there would otherwise be no oscillation.

This dilemma between small and large step sizes for the back-propagation algorithm creates a need for empirically determining a “good” learning rate for the problem in hand. Manausa and Lacher (1993) suggest that since it is the derivative of the activation function that causes oscillation in the procedure, we should limit the value of the learning rate parameter to a maximum of two. The reason for this, they claim, is that the derivative of the activation function exhibits initial condition sensitivity and has chaotically wandering dynamics when the learning rate parameter is greater than two. They went on to present a dynamic parameter update method to avoid oscillation.

Manausa and Lacher (1993) found that small changes in the lambda parameter could greatly alter training time and even whether training is possible. Combined with the chaotic phenomena, this means that it is impossible for simplistic recipes to guarantee convergence criteria. They concluded that perhaps a full optimisation strategy for back-propagation could be found only by optimising all the variables that compromise the system functions together.

3.4 The Problem of Local Minima

The error of an artificial neural network as function of the weights is known as the error surface. When a neural network is trained on a problem without hidden units, the error surface is guaranteed to be bowl-shaped, and so gradient descent is guaranteed to find the global minimum of the error surface (even if it may be non-zero – see Rumelhart and McClelland, 1988). With hidden units, however, the error surface is not guaranteed to be a simple bowl-shape and there is a danger of getting stuck at one or more local minima of the overall error function.

Figure 4 shows the problems beset by a point undergoing steepest gradient descent in the weight-space for some hypothetical problem where the point starts out near a local error minimum. This can be easily visualised by a ball rolling down the error surface and coming to rest at one of the minima of overall sum squared error, *E*. The ball represents a point in weight-space undergoing steepest gradient descent.

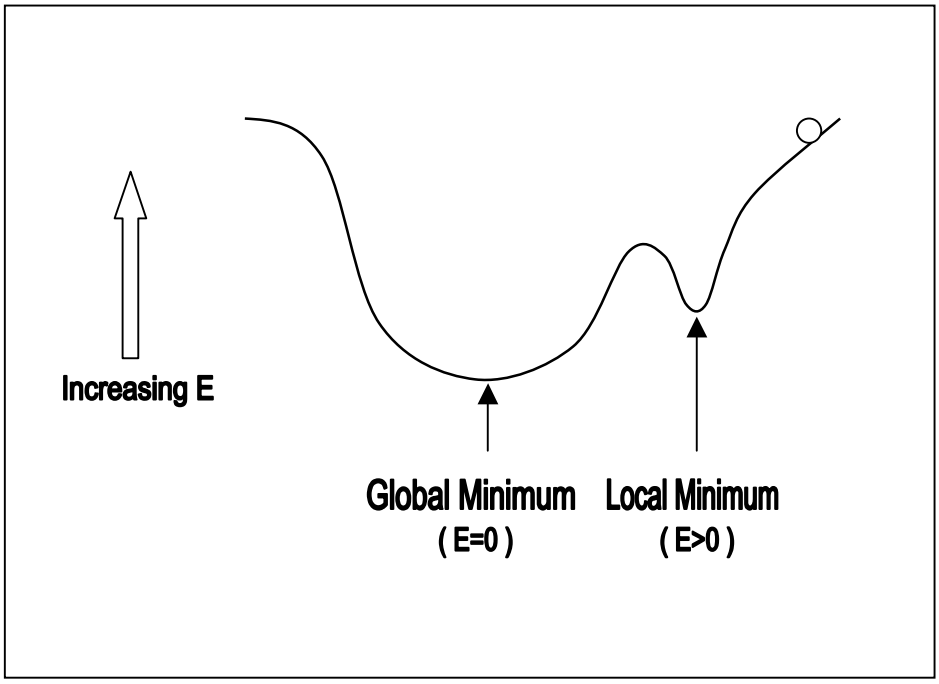


Figure 4. Hypothetical error surface showing local and global error minima.

The gradient descent procedure comes to a rest at points of the error surface where the gradient of the error is zero, and such points are referred to as the minima of the error surface. The point of true lowest error in the diagram, the global error minimum, attracts points within its vicinity, and once in this vicinity a point has no way of escaping. The point eventually comes to a rest with an overall sum-squared error of zero. Moreover, when the sum squared error reaches zero, no further weight changes can be made. This feature is independent of the learning rate parameter setting. The local minimum in the diagram similarly attracts points within its vicinity, but a point is not guaranteed to stop there since it will have some fed-back error so that the weights can still possibly change.

Sprinkhuizen-Kuyper and Boers (1996a) make a useful distinction regarding the stability of error minima. They define a stable minimum as one that contains only points with a gradient equal to zero for the error of all training patterns individually. Unstable minima will have a zero gradient for the summed error of the total set of training patterns, but not for the error of the patterns individually. They make the interesting point that "the fact that all local minima are unstable can be exploited by the learning algorithm to escape from these minima" (Sprinkhuizen-Kuyper and Boers, 1996a, p. 1319).

The generalised delta rule can also get stuck where a unit is numerically saturated. The numerical saturation occurs as soon as the net input of a unit results in the value of the squashing function being very close to 0 or 1, since the derivative of the squashing function approaches zero for these output values.

Many researchers have investigated the local minima problem using numerical methods (see for example Gorse, Shepherd and Taylor, 1993; Herz, Krogh and Palmer, 1991; Lisboa and Perantonis, 1991). Gorse, Shepherd and Taylor (1993) claim that on-line learning with a reasonably large learning rate parameter is best for avoiding local minima.

Rumelhart, Hinton and Williams (1986a,b) made some very confident statements about the optimal convergence of the generalised delta rule, arguing that although local minima did exist, they were "not significantly worse than the global minima" and "irrelevant in a wide variety of learning tasks." (Rumelhart, Hinton and Williams, 1986b). In summary, Rumelhart, Hinton and Williams (1986a) add, "We do not know the frequency of such local minima, but our experience with this and other problems is that they are quite rare." These kinds of subjective statements are probably the weakest link in all the threads of the specification of the generalised delta rule.

In more accurate accounts, Rumelhart and McClelland (1986a) stated that they had taught generalised delta rule systems to solve the exclusive-or problem hundreds of times, and that in only two cases had the system encountered a local minimum. Both cases involved the 2:2:1 version of the exclusive-or problem (see Figure 3) and both cases ended up in the same local minimum.

The local minimum problem is a very significant failing of the generalised delta rule. On one hand, it is claimed (Rumelhart, Hinton and Williams, 1986a,b) that there is a powerful neural algorithm that can, in principle, solve virtually any computational problem when defined over input/output exemplars. But then, as to the problems of the network getting stuck in a solution state that is not correct, the authors claim irrelevant, and only apparently fatal.

The generalised delta rule is a truly parallel artificial neural learning procedure, but the procedure can fail with respect to guaranteeing convergence to the global minimum of the sum squared error of the system. This poses a real problem because, for example, if we build a large scale network to solve a serious problem we do not want to have to put the system through possibly millions of training iterations only to find that it has converged to a solution that is less than adequate. This point seems to make a mockery of the great claims for the generalised delta rule procedure.

Because of this weakness in the specification of the procedure, many researchers after Rumelhart, Hinton and Williams (1986a,b) proposed a plethora of 'improved' versions of the procedure. Indeed, "a vast variety of improvements" (Riedmiller and Braun, 1993) of the technique for training the weights in a feed-forward neural network have been proposed.

The problem of escaping from local minima may be analogous to the quantum physics example of finding the correct potential energy required to overcome or tunnel through a potential barrier, or escape from a potential well, on the error-surface. Taking the error surface to be analogous to a kind of thermodynamic landscape, we might even be able to use dynamically ergodic learning trajectories to escape some minima. Some models operate in a manner conceptually very similar to this idea, such as Boltzmann machines (Hinton and Sejnowski, 1986) and Harmony nets (Smolensky, 1986), which both use simulated annealing to escape from local minima. If the annealing schedule is slow enough, a global minimum is found, but such a schedule is hard to find and therefore, in practice, these networks are not guaranteed to find a global minimum even in exponential time (Pinkas and Dechter, 1995).

One promising dynamical property of the generalised delta rule is that it is stable with respect to the lambda parameter when in an error minimum of zero, so when the system is operating with zero error the weights will remain stable no matter what the lambda parameter is. In addition, if a network has sufficient weights to represent a problem with an error of zero, then a value or range of values for the lambda parameter might be found that lead to convergence to a global error minimum irrespective of other system parameters.

Using this framework, the existence of a global solution where all initial conditions converge to a global error minimum comes about purely because of the very definition of the generalised delta rule, considering the effect of the lambda parameter upon the dynamics of the error-surface. Rumelhart, Hinton and Williams (1986a,b) summarised that, in general, the best way to avoid local minima is to use very small values of the lambda parameter, or to add more connections. In this study, I shall empirically prove that the first of these claims could be mistaken.

Chapter 4

Dynamical Mappings

The generalised delta rule is an iterative procedure governed by non-linear equations, and is thus deterministic. However, the rule is known to exhibit sensitivity to initial conditions and to have chaotically wandering dynamics, characterised by an unpredictable time evolution (Manausa and Lacher, 1993). The irregular and unpredictable time evolution of many non-linear systems has been dubbed *chaos*. The dynamical phenomenon of chaos was first named as such by James Yorke in 1975 (Yorke and Li, 1975; Gleick, 1987).

A central characteristic of chaos is that the system does not repeat its past behaviour. Yet, despite the lack of regularity, chaotic dynamical systems are governed by deterministic, non-linear equations. The condition that the equations governing the system be non-linear does not guarantee chaos; rather make its existence possible. The irregularity is part of the intrinsic dynamics of the system. The fact that deterministic non-linear rules do not imply either regular behaviour or predictability has had a very pervasive impact on many fields of science.

In this chapter I will attempt to find methods to determine the dynamical behaviour of the generalised delta rule. The determination of the dynamics of the generalised delta rule is crucial in understanding the long-term behaviour of systems that employ this algorithm. Finding tools or methods that can determine stability in amongst the chaotic dynamical phenomena found in generalised delta rule systems could gain an important insight into the long-term behaviour for such systems.

4.1 Looking for Stability in Dynamic Systems

Chaos can often be mistaken for random noise. We know that simple but non-linear rules can have seemingly random behaviours. While chaotic behaviour does appear to be random behaviour, it is actually predictable over short periods of time (Hunt and Johnson, 1993). One early theory concerning the mechanism or route to turbulence was proposed by Landau in 1941 (see Feigenbaum, 1980). The idea was that a system becomes turbulent through a succession of instabilities, with each new instability creating a new resonant mode for a given system.

Most modern computer languages are equipped with pseudo-random number generators. Many algorithms for the random number generation actually generate a repetitive sequence of numbers over a repeating block that is very long. Accordingly, it is possible to predict what the *nth* generated number will be. And yet, the generated number sequence may be subjected to statistical tests to show that all combinations of numbers occur equally often, and so on (Lauwerier, 1991). The sequence is determined but gives the impression of being chaotic, or random. Technically, the term pseudo-random is used to indicate this nature.

One can produce pseudo-random number sequences with non-linear iteration. For some non-linear iteration schemes, the sequence of numbers generated for all starting points within a given range can possess *all* the mathematical properties of a random sequence (Feigenbaum, 1980). These systems can model many complex phenomena, such as the behaviour of a population from generation to generation, turbulent flow in a fluid, etc.

Non-linear iterative systems rely on recurrence relations. We can relate the next value (the iterate) of x_n, x_{n+1}, by some function, f:

$$x_{n+1} = f(x_n) \qquad (8)$$

Such iteration schemes are trivial to implement on a digital computer, given that we disregard errors from the digital floating-point representation etc. If we are interested in the long-term behaviour of the system then we must find ways of predicting the value of the *nth* iterate after much iteration of (8). When the iterated function *f* is linear, an analytical solution is in general possible, but when *f* is non-linear, the value of the *nth* iterate x_n as a function of the initial value x_0 can be unavailable.

Feigenbaum (1980) was one of many who were interested by the fact that complex, seemingly random events from nature, (such as chaotic flow in a fluid, weather patterns, thermal noise in electronic systems, etc.) might be merely pseudo-random, and not truly random: "which is untestable, a historic but misleading concept" (Feigenbaum, 1980). This, I believe, is a profound insight - that our very concept of what randomness means is possibly mistaken.

This concept at first appears somewhat confusing, but we can find examples of this phenomenon if we look back over time. A few decades ago, for example, when analogue electronics were used to make computing machinery, many methods were used to create white noise signals for processing and creating random number sequences. One such method was to tune a circuit to pick up the microwave background radiation, to amplify that signal, and use it as a white noise source. It is only in recent years that we have discovered that the microwave background radiation is not actually random - it has a large-scale rippled structure that was previously unknown (Smoot and Davidson, 1993). This is an example of how we might easily call something random when what we really mean is that we do not yet understand the mechanism behind it. Another way of expressing this concept is to say that if we can find no law to predict a certain behaviour, we might label it as 'random' – until we do in fact discover a predictive law for it.

4.2 Simple Bifurcation Map

To start with, let us specifically consider the logistic recurrence relation, known for its chaotic behaviour (Bertels, Neuberg, Vassiliadis and Pechanek, 1995):

$$x_{n+1} = \lambda x_n (1 - x_n) \qquad (9)$$

This relation is sometimes called the logistic map, and is a quadratic map. With $\lambda = 4$, the logistic map has a maximum value of 1 at $x_n = ½$, and the values of x_n from 0 to ½ are mapped from 0 to 1. Similarly the values of x_n from ½ to 1 are mapped from 0 to 1, but in reverse order. Both intervals of x_n are stretched by a factor of 2, but because the order of the mappings is opposite, the second stretched interval is folded onto the first stretched interval. In this way the logistic map is said to be folded.

Because the logistic map is folded, there is always ambiguity in trying to retrodict x_n from x_{n+1}, and so the logistic map is said to be noninvertible. The same noninvertability is found with elementary functions such as $y = \sin x$. The inverse functions can be defined only by limiting the original domains. The ability for the logistic map to have complex behaviours is precisely the consequence of its double-valued inverse, which in turn is a consequence of it being a folded function.

We are interested in the long-term behaviour of the repeated iteration of equation (9). Figure 5 shows a bifurcation diagram or scatter plot of the long-term values of x over a range of values of the parameter λ. This diagram was constructed by choosing initial pseudo-random values for x of between 0.1 and 0.9, and then applying equation (9) for one thousand iterations. For each different value of the λ parameter, we test 100 different initial conditions. Only the last value of x is plotted.

The bifurcation diagram allows the dynamics of a non-linear system to be viewed globally over a range of parameter values, thereby allowing simultaneous comparison of periodic and chaotic behaviour. The diagram provides a summary of the essential dynamics and is therefore a useful method of acquiring this overview. The bifurcation diagram is thus an important tool for discovering interesting parameter regimes for a dynamical system.

The bifurcation diagram shown in Figure 5 is very complex. For certain ranges of the λ parameter, x takes on an infinite number of different long-term values, though there are also many holes. It is also interesting to note that as the λ parameter is varied there are small intervals in which the behaviour abruptly changes, becoming either periodic or non-periodic.

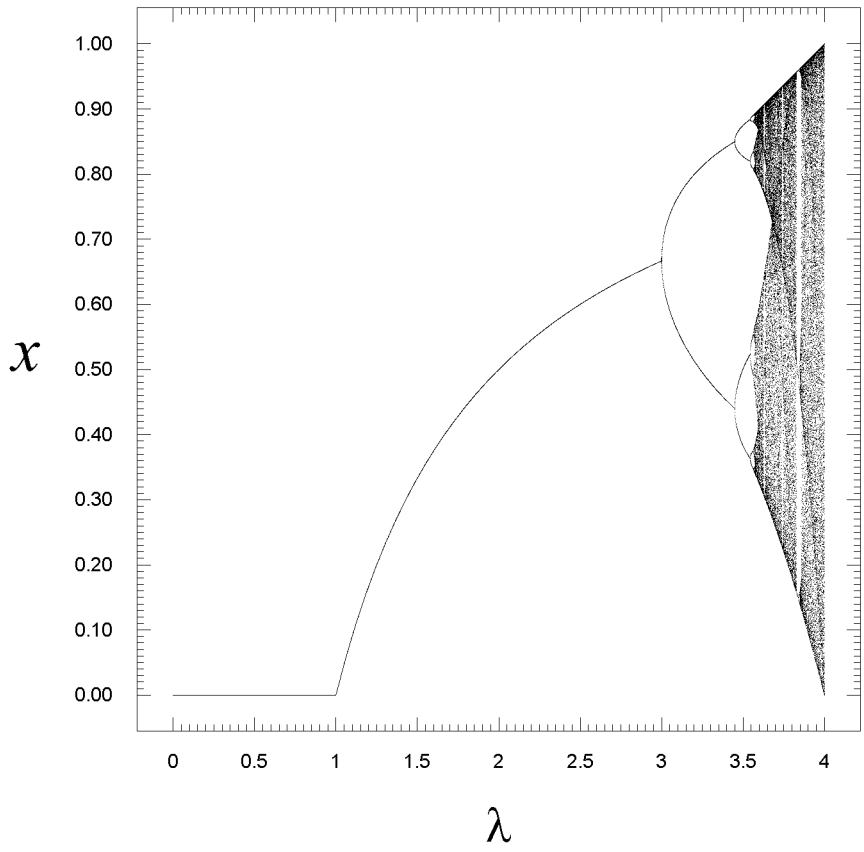


Figure 5. Bifurcation diagram showing the 1,000th iterate of the logistic function $\lambda x(1-x)$, plotted against λ. Initial values of x of between 0.1 and 0.9.

One way to begin to deal with the problem of predicting the *nth* iterate is to consider special cases. One such special case would be where an initial value for x_0 is mapped, via *f*, onto itself, producing $f(x_0) = x_0$. Therefore, all applications of the function *f* will produce x_0, and the sequential behaviour of the system is then $x_0, x_0, x_0, \ldots$ Cases where $x_{n+1} = x_n$ are referred to as the fixed points of *f*. A stable fixed point where many initial conditions converge is an attractor. The limiting behaviour of the iterates of *f* is determined by the stable fixed points of *f*, the attractors. This limiting behaviour is independent of the initial starting point x_0 provided that x_0 is within the basin of attraction of the attractor. A non-linear system may have a population of such attractors.

After choosing the initial value for *x*, and iterating many times, we will find that we arrive at a limiting value of x_n, and that the limit value can be calculated. When $0 < \lambda < 1$, the value tends towards zero, or $x_n \rightarrow 0$. When $1 < \lambda < 3$, the limit value can be calculated as $x_n \rightarrow 1 - (1/\lambda)$. Iteration of *f* will eventually reach a fixed stable point with a value always greater than zero.

The fixed point condition for *f* where $x=0$ is always a fixed point, even if λ is varied. The other fixed point $x = 1-(1/\lambda)$ becomes available when we increase λ past 1. When we increase λ to 2, the fixed point is determined by $x = 1-(1/2) = ½$. No matter how close the initial value x_0 is to the fixed point at $x=0$, future iterates will diverge away from it and towards the fixed point at $x = ½$. Thus the fixed point at $x=0$ is termed unstable, and that at $x = ½$ is termed stable (see, for example, Ditto, Spano, and Lindner (1995) for definitions of these fixed point behaviours).

In dynamics, a change in the number of solutions to a differential equation (or a system of difference equations) as a parameter is varied is called a bifurcation (the Latin *furca* means fork). It causes the system to cycle between two values of x with the exact value dependent on the initial condition. For example, we could generate the sequence $x_0, x_1, x_0, x_1, \ldots$. Taking each iteration n as a discrete time step, the system's behaviour reproduces itself every period of time $2n$. If we view the behaviour of the system as periodic then sequences such as this that cycle between two values have a period of two. Moreover, the period of the sequence has doubled as a parameter has been increased past the range where the behaviour was stable.

We can see from Figure 5 that when the parameter λ is increased to 3, the system undergoes period doubling, whereby a 2-cycle or attractor of period 2 emerges (Feigenbaum, 1980; Baker and Gollub, 1990). The system oscillates between two values, so $x_{n+2} = x_n$. The two points of this limit cycle are not fixed points of f, rather they are fixed points of f^2, the function f applied twice (Feigenbaum, 1980).

When the parameter λ is further increased to $1+\sqrt{6}$, a limiting cycle of 4 values appears (a very interesting discussion of this mechanism appears in Lauwerier, 1991). The behaviour requires $4n$ for reproduction, so $x_{n+4} = x_n$. Again, the period of the sequence has doubled. This time, we have a limit cycle of 4, comprised of the four fixed points of f^4, the function f applied four times.

The values of λ for which the transitions occur are called bifurcation points, the transitions are called bifurcations. The successive doubling of the period of each further limit cycle is called period doubling. This process of successive period doubling recurs continually until, at a certain critical parameter value, it has doubled ad infinitum, and the behaviour is no longer periodic, in the sense that we would require an infinite number of iterations for the sequence to repeat itself. This behaviour occurs at around $\lambda_\infty = 3.56994\ldots$ (Baker and Gollub, 1990; Lauwerier, 1991). The behaviour is now chaotic and "similar to that of a random process with a uniform probability distribution" (Baker and Gollub, 1990).

Between λ_∞ and $\lambda = 4$ there is a denumerably infinite number of λ values for which the system behaves periodically, and an indenumerable number of λ values for which the system behaves chaotically. There is an interval near $\lambda = 3.83$ where a stable 3-cycle occurs, called a window of periodicity. The third order map f^3 near $\lambda = 3.83$ has three corresponding fixed point attractors. As λ is increased there is another period doubling cascade to 6-, 12-cycle behaviour etc.

4.3 Universality Theory

Universality theory (Feigenbaum, 1980) expresses the idea that certain universal numbers determine quantitatively the transition from smooth to turbulent behaviour for a large class of non-linear systems. Universality theory is just one of a vast array of methods to be found under the umbrella of chaos theory.

The period doubling mechanism is one route to chaos, achievable whenever a system is governed by non-linear functional iteration. There is a unique and hence universal solution common to all systems undergoing period doubling (Feigenbaum, 1980).

The period doubling route is particularly interesting because it may be characterised by certain universal numbers that do not depend, to within certain limits, on the nature of the map. For example, the ratio of the spacing between consecutive values of λ for each successive bifurcation approaches a constant called the Feigenbaum number, after its discoverer. If the first bifurcation occurs at λ_1, the second at λ_2, and so forth, then this universal number is defined as:

$$F \equiv \frac{\lambda_{n+1} - \lambda_n}{\lambda_{n+2} - \lambda_{n+1}} \equiv 4.6692016091029909... \qquad (10)$$

The Feigenbaum number F is the rate of onset of such complex behaviour. It is a predetermined, universal constant. This number can be roughly checked by careful scrutiny of the bifurcation diagram in Figure 5. Knowing this constant, and having measurements of at least two period doubling points from the bifurcation diagram, allows us to predict values of λ_n for all the other period doubling cycles.

The Feigenbaum number is a universal property of the period - doubling route to chaos for iterations of functions that have a quadratic maximum, such as the derivative of the activation function used by the generalised delta rule. Universality theory expresses the notion that high iterates of such functions have a certain geometry that is independent of the specific form of the function. It therefore follows that any universality effects found at high iterates of the generalised delta rule will be independent of the actual form of the activation function.

If we take a system where neighbouring regions of initial conditions converge to a specific region, such regions are called basins of attraction for that system. A basic theme of universality theory is that attractors make the eventual behaviour of the system independent of the starting point. This concept is crucial to understanding the long-term behaviour of the wandering nature and the sensitivity to initial conditions found in many non-linear iterated systems.

Universality theory is a difficult concept. Strangely, we are helped in our analysis because we are reapplying the same function, and the long-term behaviour can be determined by the key notion of iteration itself, irrespective of the characteristics of the particular function used for the iteration.

Systems of differential or difference equations naturally determine certain maps. The computation of the analytic form of the map is generally very difficult using standard mathematical methods. However, should the map exhibit bifurcation effects, then precise predictions are available from universality theory. These predictions apply independently of which specific map it happens to be. Under universality theory, dimensionality is also irrelevant. The same geometrical property quantitatively determined by the number *F* will work for iterations in *N* dimensions, provided that the system goes through period doubling.

4.4 Possible Applications to Generalised Delta Rule Systems

A very important characteristic of generalised delta rule schemes is that the equations that govern them are non-linear. A consequence of this is that, although these systems are deterministic, their behaviour is not necessarily predictable, and they can exhibit unstable behaviour such as parameter sensitivity and sensitivity to initial conditions (Baker and Gollub, 1990; Pollack, 1990). Learning algorithms that are non-linear can exhibit a seemingly random wandering nature.

The generalised delta rule is a non-linear iterative procedure. A feature common to all non-linear iterative systems is that as some parameter of a function is varied (temperature, for example), the behaviour changes from simple to erratic (Gleick, 1987). For low parameter values, the system is stable or “smooth”; it does not periodically oscillate. For a high parameter value, the system can become periodic and oscillate. The system may oscillate wildly, and then settle on a solution. This study will focus on clarifying the mechanisms behind this dynamical, wandering nature in systems that use the generalised delta rule.

Measuring the dynamics of non-linear artificial neural learning systems such as the generalised delta rule is important primarily because the dynamics are so poorly understood. Moreover, the generalised delta rule is poorly specified with respect to the fitness of convergence, and other properties. The goal is therefore to achieve a clear understanding of the dynamical properties of generalised delta rule systems, and by understanding these properties, to then develop new ideas for further advancement in this field.

Universality effects may have important consequences for the convergence properties of the standard generalised delta rule. In the standard procedure, the derivative of the sigmoid activation function has the form $O\ (1 - O)$, and the learning rate parameter λ is used to scale each individual weight increment. Given the similarities between this relation and the form of the logistic map (Equation 9), it would seem natural to observe similar phenomena such as bifurcations and chaos as found was found for the logistic map (Bertels, Neuberg, Vassiliadis and Pechanek, 1995).

We can thus reasonably expect that the generalised delta rule will be significantly affected by altering the learning rate parameter λ, much the same as when the simple logistic function is iterated. Indeed, we can expect that the different long-term behaviours for the different λ values should scale to any generalised delta rule system, being independent of the specific form of the derivative of the activation function (the error function), as long as it is a folded function. The different behaviours induced by the value of the parameter λ should affect the behaviour of the error function, in a manner that may well be predictable.

Now that we have deduced that it is the error feedback component that may force the system to undergo dynamical changes as a result of the manipulation of the λ parameter, we are led to the conclusion that in order to study the dynamical behaviour of a generalised delta rule system we should employ a scatter plot similar to a bifurcation diagram, with λ as the independent variable, and use a unit corresponding to the limiting error of the system for the dependant variable.

Applying a scatter plot or bifurcation diagram to an artificial neural network has been suggested before (see for example Renals, 1990; Bertels, Neuberg, Vassiliadis and Pechanek, 1995, 1998) but all of these investigations only studied the output values for the neural network, in order to determine whether chaos would be present or not. This study specifically proposes the measurement of the *error* of the output values, to determine convergence properties of the network.

For any network utilising the gradient descent method, bifurcations in the number of limiting solutions can be easily detected by examining a graph of the eventual error of the network sampled after a certain number of learning trials, versus the learning rate parameter λ. After going through an initial transient, a system may come to a stable fixed point. This may take many iterations, especially if the system is degenerate. The number of fixed points is dependent on the complexity of the error surface in terms of both its form and scaling. For some values of the λ parameter, a network will have only one long-term solution, while for slightly different choices, two or more solutions may be possible. If several solutions are stable, the actual behaviour will depend upon the initial conditions.

Under the universality theory proposed by Feigenbaum (1980), dimensionality is irrelevant, provided that we find period doubling behaviour. This is fortuitous for the study of generalised delta rule systems because it means we can apply the theory irrespective of the number of weights. Provided that we find period doubling behaviour for some values of λ, we can safely predict limiting behaviour for other values of λ by using the number F.

In this study I will propose empirical methods to determine regions of optimal dynamical performance, and also conjecture that the generalised delta rule does have an empirically demonstrable, perceptron-like convergence proof for Boolean algebraic functions. The demonstration is not an analytical proof, but the methods proposed do allow regions of optimal dynamical performance to be determined, and even in the worst case allow selectable odds of success to be measured.

Chapter 5

Pilot Study

5.1 An Example with Known Local Minimum Problems

The basic 1:1:1 topology of Figure 1, when trained on the unary Boolean functions, provides a very simple example of local minima phenomena. The Boolean identity function, where an input/output transfer function of $0 \Rightarrow 0$ and $1 \Rightarrow 1$ is required, gives us a simple starting point for our experimental purposes. This problem has been extensively studied by Rumelhart and McClelland (1988) who report that, when using a fully biased 1:1:1 network, there are two solutions. The network can either settle into a state where both the hidden and the output units function as inverters (a double inversion producing identity) or both function as followers. Either of these configurations will solve the identity function with, eventually, zero error, and thus there are two symmetrical global error minima for a biased 1:1:1 network when trained on the identity function.

The error surface for the 1:1:1 network without biases is plotted in Figure 6. Without the bias terms, there are no global minima, but instead two local minima. These minima are asymmetrical, with one solution having the greater error. The system state can be thought of as being drawn towards two basins of attraction, one at a different height to the other. The solution having the greater error lies in the upper part of the plot, and the solution with the lowest error is found at the bottom right.

This simple plot is possible since there are only two weights. The first weight, w_1, is that connecting the input to the hidden unit and the second, w_2, is that connecting the hidden unit to the output unit. The target values are set at precisely 1.0 and 0.0 for true and false, giving infinite weight solutions.

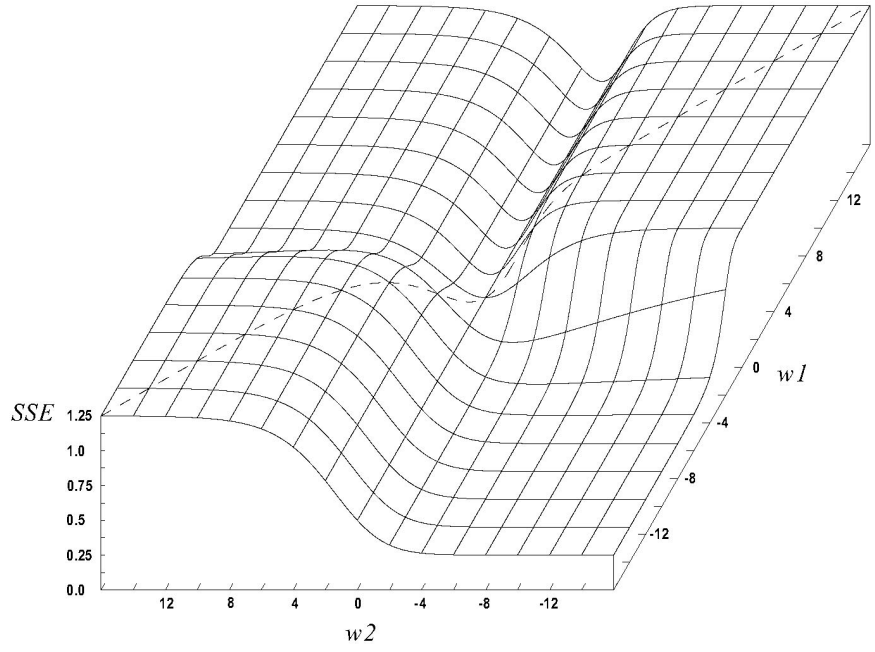


Figure 6. The error surface of a 1:1:1 network without biases trained on the identity function.

We can see that a basin of attraction representing the local minimum with the greater error, that of *S.S.E.* = 0.5, occurs along a line where $w_1 > 0$ and $w_2 = 0$. The basin of attraction found at the bottom right approaches a minimum error of *S.S.E.* = 0.25 as w_1 and w_2 both tend to minus infinity.

The point on the separatrix at the centre of the error surface, where both weights are zero, is a stationary point, because the gradient at this point is zero. This point is also a saddle point, as it is bounded by regions where both points with larger error values and with smaller error values can be found.

The type of visualisation shown in Figure 6 is fine for networks that are limited to just two weights, but we cannot produce a similar plot, contour map, or otherwise, to visualise the error surface for larger nets- for example, a 1:1:1 network with biases. This would require a four-dimensional space to represent all possible weight configurations -- clearly, as the complexity of the network increases, the direct visualisation of the error surface becomes increasingly impractical. Chang and Mak (1999) have also noted the requirement for a plot of error against the weight values as necessary for the evaluation of gradient learning algorithms, and the difficulty of achieving this for higher dimensional weight spaces. In the next section I will propose a method by which a bifurcation diagram can be used as a tool for the study and evaluation of the dynamics of the error surface for networks of arbitrary dimensionality.

5.2 Bifurcation Maps for the 1:1:1 net Learning Identity

The bifurcation mapping method used here is very simple. All of the weights are initialised to random values, the network is the trained for a certain number of iterations, and the resulting error of the network is measured at the last iteration. The error is then plotted against the value of the learning rate parameter λ. This is repeated for a number of different initial weight values for each value of λ, and a range of λ values are tested. The final error values are converted into a measure of percentage correct, and so the performance is defined by *100 * (1-abs(mean error))* and denoted as ‘‘percentage correct’’. Correct output states of zero or one for all presentations will thus convert to 100% correct, output states of 0.75 for target states of 1.0 will convert to 75%, etc. In theory, the points should cluster along horizontal lines showing the number of stable solutions for a given parameter range. In all of the experiments in this pilot study, the on-line training regime is used. All of the experiments in this pilot study were coded in Lisp and implemented on Sun Sparc computers.

Typically, a bifurcation diagram is taken after a large number of iterations, and the initial values are discarded. This ensures that we see only the long term behaviour and ignore any transient behaviour. For example, Bertels, Neuberg, Vassiliadis and Pechanek (1995) have produced two bifurcation diagrams for the XOR problem by training the network for 40,000 and 500,000 iterations. In each case they kept a record of the last 200 output values, and plotted these output values against different values of the λ parameter. In a similar study, Renals (1990) produced a bifurcation diagram by training a network for 20,000 iterations, and keeping the last 10,000 results to be subsequently plotted. Both studies made no record of the final error, and in both studies only one initial condition was tested for each different value of the λ parameter.

In this study, I shall be concerned with the eventual error of the learning algorithm at a specific and often relatively low number of iterations, and across a range of initial conditions. For example, Figure 7 shows the final error for a 1:1:1 network without biases trained on the identity function for 100 iterations, given a range of initial conditions for each different value of λ. A standard bifurcation diagram would not apply here because at 100 iterations we do not have enough data points to discard. Also, we are only interested in the error after a specific number of iterations, and wish to exclude the error measured at previous iterations.

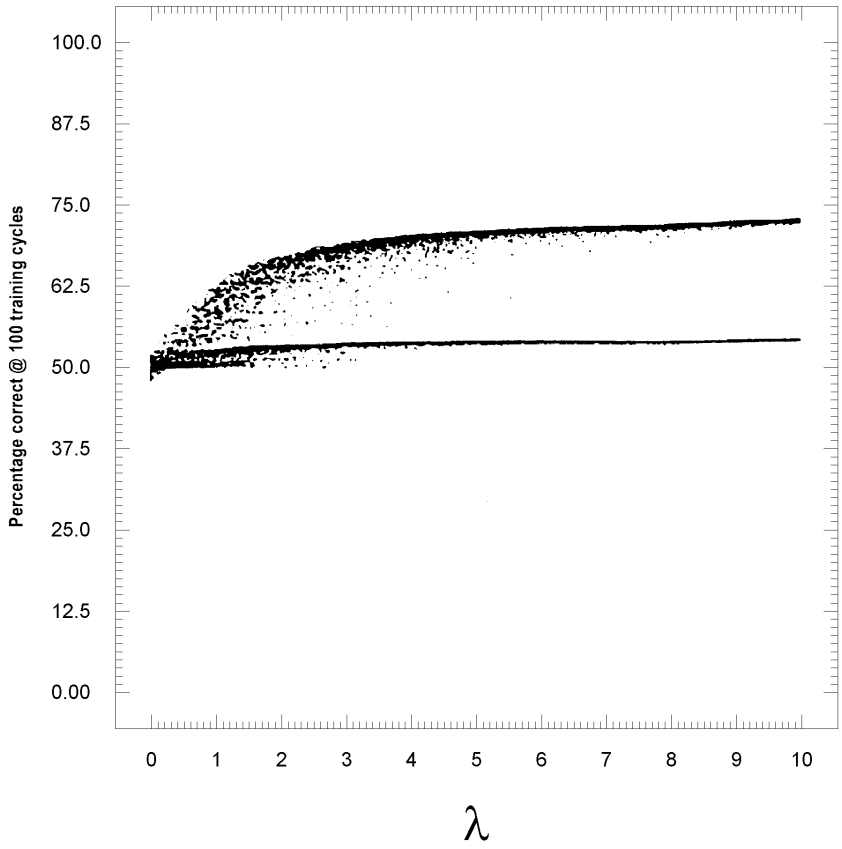


Figure 7. A bifurcation diagram of a 1:1:1 network without biases trained on the identity function. Shows percentage correct after 100 training cycles plotted against λ, which is varied from 0 to 10.

Figure 7 shows a bifurcation mapping for a 1:1:1 network learning the identity problem without biases. The diagram shows the percentage correct after 100 training cycles, plotted against the λ parameter. This plot clearly shows the two different error minima solutions for the same problem as illustrated in the error surface plot of Figure 6. To get a good range of the possible stable solutions, each value of λ was tested 10 times, each with different random initial weight values. A relatively large initial weight condition range was selected, with $-10 < w_x < +10$.

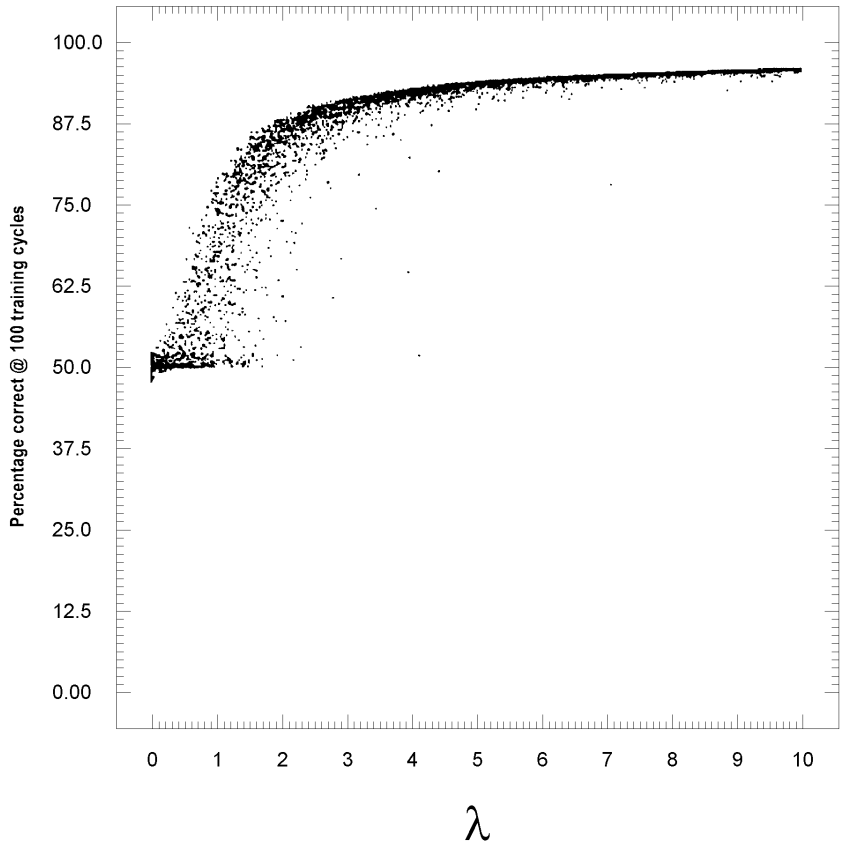


Figure 8. A bifurcation diagram of a fully biased 1:1:1 network trained on the identity function. Shows percentage correct after 100 training cycles plotted against λ, which is varied from 0 to 10.

Comparing Figures 7 and 8 one can see that the network will, at best, settle to the nearest error minimum from the given starting conditions. Figure 7 clearly shows two rough convergence lines, one the global minimum and the other the local minimum. If there is only one global minimum, and global over all initial conditions, then we would expect to see a single convergence line in the bifurcation diagram. This is similar to the case for the problem shown in Figure 8, which shows a bifurcation diagram for the 1:1:1 identity problem using biases (four weights, in total) - recall that there are two symmetrical global minima for this problem, which in the limit converge to two possible weight configurations, with both having equal error. The two solutions both converge to the same region in the diagram.

One of the aims of this study is to demonstrate convergence to global error minima using the lowest number of training iterations. Because we are limited to testing at a relatively low number of training iterations, these diagrams appear to be very fuzzy, with many of the points showing incomplete convergence, especially at the regions where we have both a low number of training iterations and low values of λ.

These diagrams can be distinguished from learning curves, which in general show the percentage correct as the number of training iterations is varied. In the strictest sense, these diagrams should be called scatter plots, but throughout this study I shall refer to these plots as bifurcation diagrams, since they do show the dynamical behaviour and bifurcation phenomena as the λ parameter is varied. Other investigators (see for example Someya, Shinozaki and Sekine, 1999) do employ similar bifurcation diagrams to study the chaotic dynamics of neural models, even though the diagrams appear fuzzy through incomplete convergence.

5.3 Bifurcation Maps for the 2:2:1 net Learning Exclusive-or

A classic example of the unpredictability of the generalised delta rule occurs when we attempt to train the limited 2:2:1 network (as shown in Figure 3) on the XOR function. The input/output transfer function for exclusive-or is 00⇒0, 01⇒1, 10⇒1 and 11⇒0. The network can, it appears, get stuck in a number of local error minima, but the necessary conditions and system parameters which lead to these minima seem to be, apparently, unknown.

The learning rate was varied from zero to 50, a rather high range, and the initial conditions were again chosen to be relatively large, with $-10 < w_x < +10$. The error of the system was sampled after 1,000 presentations of the four input patterns.

Because in this section we are using the on-line training regime, whereby the weight updates are applied after each presentation of each input pattern, we can easily study the effect of the input pattern presentation order. A number of bifurcation diagrams were therefore computed for the 2:2:1 network learning the exclusive-or function, one for each permutation of input presentation ordering. Since the exclusive-or function is commutative, we would not expect a 2:2:1 network learning the exclusive-or function to exhibit handedness, so all input/output presentation orderings with input patterns 01 and 10 reversed were excluded.

Upon examination of the resulting bifurcation diagrams, it was found that the diagrams fell into two categories, with each resulting category sensitive to the presentation order of the input patterns. The results were therefore grouped together, according to which of the two categories the resulting diagrams fitted.

Figure 9 shows a bifurcation map for the first category, which overlays the results from testing all input pattern presentation orderings except 00.01.10.11, 01.10.11.00, 01.00.10.11 and 01.10.00.11. Given these conditions, the diagram shows that there are three main error minima for the 2:2:1 network learning the exclusive-or function, one global and the other two local.

At very low values of λ, near zero, the algorithm makes very little progress from the initial conditions, and the percentage correct performance is between 50 – 100%. As λ is increased, within the range $2 < \lambda < 25$, we find that the errors converge to 3 distinct bands, representing the 3 minima for the problem. As λ is increased past 25, a new band representing 50% correct performance appears, and the two previous lowest minima bands begin to merge together. At $\lambda = 50$ we are left with distinct minima of 100%, 75% and 50% correct.

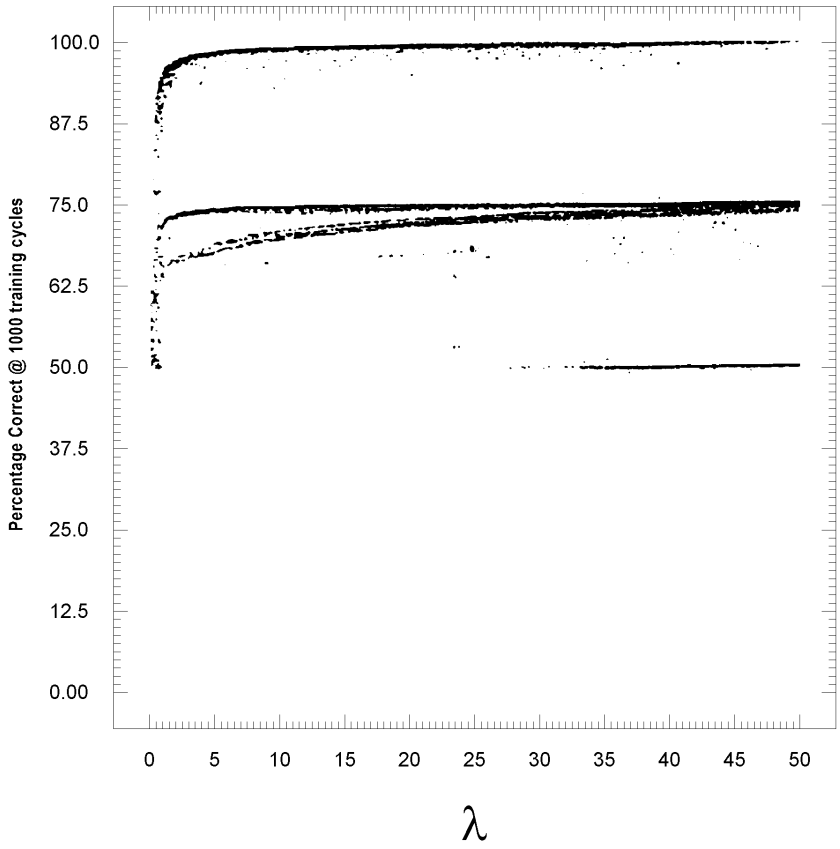


Figure 9. Bifurcation diagram for the 2:2:1 exclusive-or problem. The figure shows the percentage correct after 1,000 training cycles, plotted against λ varied 0 to 50. This map is for all input pattern presentation orderings except 00.01.10.11, 01.10.11.00, 01.00.10.11 and 01.10.00.11.

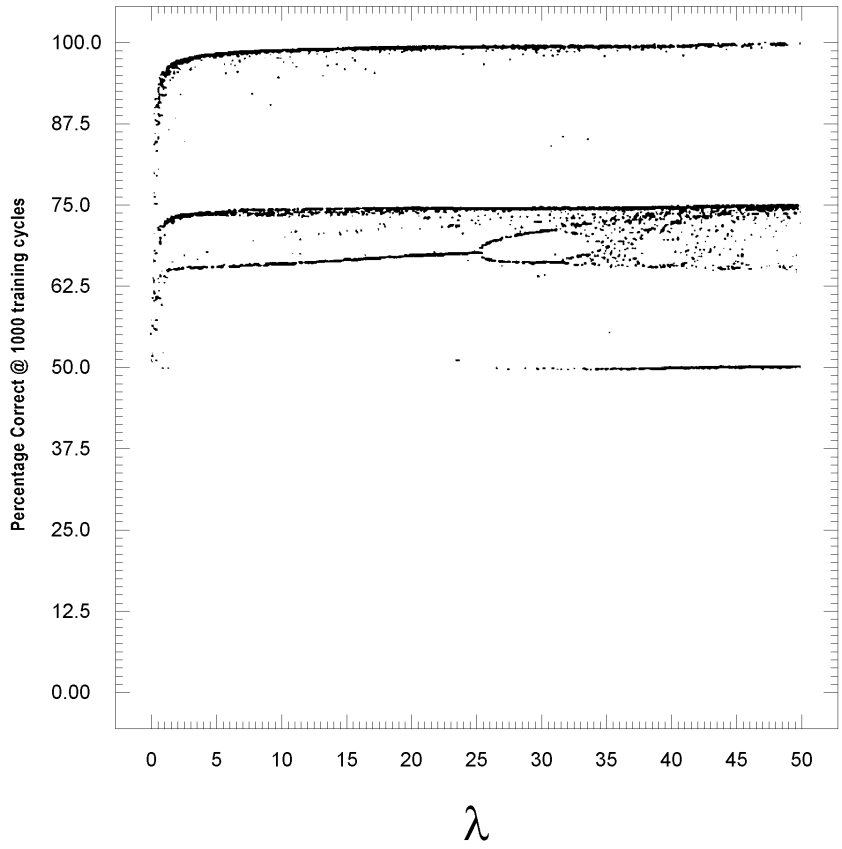


Figure 10. Bifurcation diagram for the 2:2:1 exclusive-or problem with biases. The figure shows the percentage correct after 1,000 training cycles plotted against λ varied from 0 to 50. This map is for input pattern presentation orderings 00.01.10.11, 01.10.11.00, 01.00.10.11, and 01.10.00.11.

The map for the input pattern presentation orderings excluded from Figure 9 turns out to be rather different, shown in Figure 10. We can see that the two 'best' solutions are still present, but the third has radically changed. This solution forms a line slightly lower than the corresponding line in Figure 9. Moreover, where the learning rate is increased further than around 25, the solution shows all the hallmarks of period doubling, a successive doubling in the population of attractor states.

The first doubling occurs as the learning rate is increased to around 25, and one can just discern a second bifurcation, to four attractor states, at a parameter value of around 31.5. This period doubling cascade has a regular geometric structure, and can be roughly checked as conforming to the ratio of spacing between successive lambda-values at each period-doubling point as predicted by the Feigenbaum number *F*. Thus, it is possible to make limited predictions as to the number of solutions for particular learning-rate values within this window of period doubling behaviour.

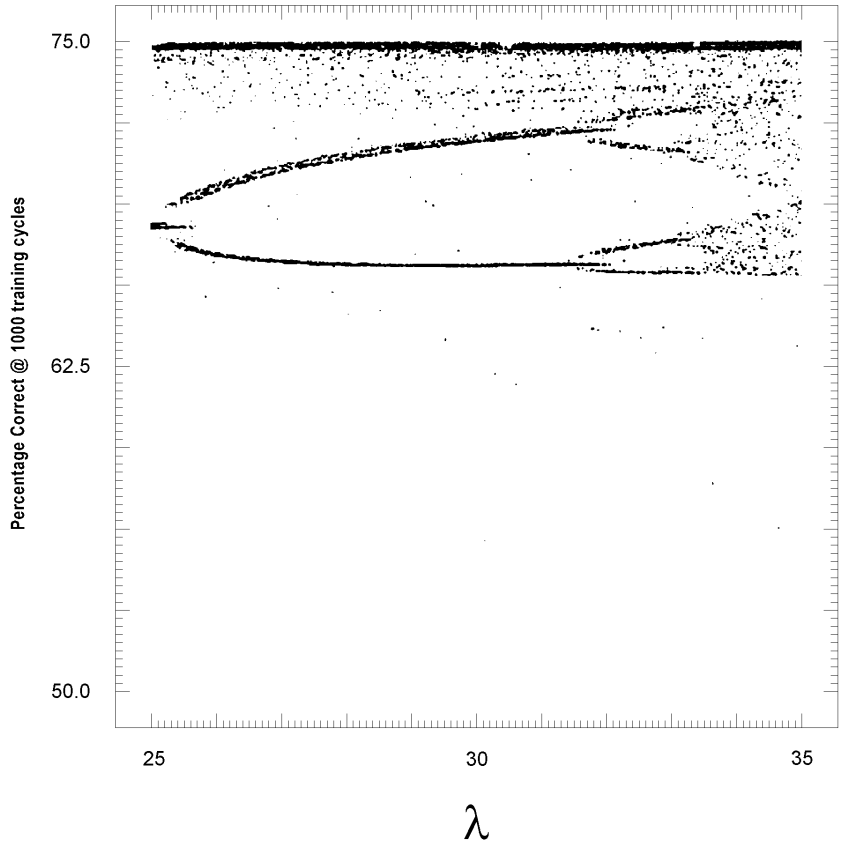


Figure 11. Magnification of period doubling region as λ is varied from 25 to 35, showing the universal period-doubling phenomenon.

5.4 Analysis

First we consider the case of the 1:1:1 network learning the Boolean identity function. For the problem tested without biases, we confirmed the two different error minima solutions as being in rough agreement with an error surface plot of the same problem, over a relatively large initial condition range. At the rather low figure of 100 training iterations the diagram appeared to be very fuzzy, with many of the points showing incomplete convergence.

The actual error surface plot (Figure 6) showed that there exits a saddle region separating the two minima. Sprinkhuizen-Kuyper and Boers (1996b) make the interesting observation that when a saddle point is encountered, batch-mode learning can get stuck in such a saddle point, but on-line learning may possibly escape from such a point, since the error surface is not horizontal for each individual pattern, only the average error surface for all patterns is horizontal. So a small weight-change in the right direction will decrease the error, moving away from the saddle point.

For this reason, Sprinkhuizen-Kuyper and Boers (1996b) claim that such saddle points should not be termed local minima, because they are theoretically “escapable”. Contrary to this, I suggest such points are true local minima, since such points are *not* escapable using batch-mode learning. Even if such points are escapable using on-line training, the input pattern presentation order may be critical for success, since one particular presentation ordering may produce the required weight-change in the right direction whereas another may not. A certain presentation ordering may “cancel out” the change in weights of the required direction, producing much the same effect as batch-mode training.

The bifurcation diagram plotted for the 1:1:1 problem without biases demonstrated that on-line training does not escape the local minima, even though it is actually a saddle region. It is, therefore, a true local minimum of the error surface considered here. For this simple example we may reason that on-line training is not guaranteed to escape from such minima, at least for all training iterations up to 100 and for the given range of λ tested. This finding would appear contrary to the claim by Sprinkhuizen-Kuyper and Boers (1996b) that saddle shaped regions are not true local minima.

For the 1:1:1 network trained on the Boolean identity function using biases, we confirmed that there can only be global minima for the problem, and no local minima, again using a relatively large initial condition range. For both topology types it was demonstrated that the network will, at best, settle to the nearest error minimum from the given starting conditions.

For the case of the 2:2:1 network learning the Boolean exclusive-or function, it was demonstrated that there are three main error minima, across a relatively large range of initial conditions, and across all input pattern presentation orderings. Once again the diagrams appeared somewhat fuzzy, most probably due to incomplete convergence. This could be due to a number of factors, such as low number of training iterations, effect of the λ parameter, and possibly the large range for the initial conditions starting a unit off near saturation point and therefore slowing down the training.

The results for the 2:2:1 exclusive-or problem also confirm that there can be definite input pattern presentation order effects in the generalised delta rule procedure when we use the on-line method of updating the weights. These effects show themselves in the presence or absence of the bifurcation phenomena at higher learning rate parameter values. The pattern presentation order effect could be avoided by using the method of updating the weights after each epoch, that is, using batch as opposed to on-line training.

Sprinkhuizen-Kuyper and Boers (1996a,b) investigate the same 2:2:1 XOR problem using the same limited-feedforward topology. They provide mathematical proofs that the error surface for this problem has a stable global minimum with error zero (Sprinkhuizen-Kuyper and Boers, 1996a) and that a number of local minima exist. They prove that these minima have boundary points that are, in fact, saddle shaped. From their analysis they conclude that the local minima occur for combinations of the weights from the inputs to the hidden units such that one or both the hidden units are near saturation (giving outputs very close to 1 or 0) for at least two pattern presentations (Sprinkhuizen-Kuyper and Boers, 1996b). This would seem the most likely explanation for the actual error minima detected in these experiments, and would explain how the input pattern presentation order effects arise. For certain training presentation orderings, the input pattern sequence may cause the saturation of one or both hidden units for the required two presentations, whilst other pattern presentation orderings may avoid this condition.

The presence of the bifurcation phenomena suggests that the procedure is definitely amenable to the universality scaling theory of Feigenbaum, in that the geometrical structure in the regions where the bifurcations occur will conform to the ratio specified by numbers such as F, and hence affords limited predictability for the procedure.

The fact that the 2:2:1 exclusive-or problem showed a break-up of the worst (lowest percentage correct) solution line under the bifurcation effects suggests that this may be a general dynamical property of the procedure that may help the system to escape from error minima. If we take a look at the diagram produced in Figure 10 we see that the lowest solution line for the 2:2:1 exclusive-or problem approaches a level of around 65% correct, and a period doubling cascade begins at a λ value of around 25. The solution line then breaks up, with the eventual percentage correct now converging to 50%, 75% and 100%. However, there is ambiguity in that we cannot determine from the diagram alone if a transition to either a higher or lower eventual error has been made. We must therefore develop a method to discern whether these dynamical transitions are beneficial in promoting lower eventual error or otherwise.

Chapter 6

A Standard Benchmark

This chapter focuses on the search for a powerful benchmarking method for the generalised delta rule system. For any powerful benchmarking procedure to be attained we must emphasise the importance of measurement, and measurement can help with predicting certain quantities. There is a saying attributed to Pythagoras: “The Unit is everything” (Wightman, 1951). Although this is an old maxim and easily dismissed, there is a deep meaning to the idea behind it. What this idea seems to be stressing is that it is not measurement itself that is the central question, what is important is our selection of a unit of measurement. Measurement should also be as general as possible, for the broadest range of application.

Reporting of benchmarks for artificial neural network performance is very unsatisfactory, with learning curves (for example see Ampazis, Perantonis and Taylor, 1999), average errors and standard deviations (see Riedmiller and Braun, 1993; Hochreiter and Schmidhuber, 1997) being useless to describe any erratic or wandering behaviour, and the relative frequencies of convergence to non-optimal minima. Researchers use different benchmarks and adopt different criteria for success of training, and a standard way of evaluating a neural learning algorithm is required.

A good benchmark should accurately measure and compare the convergence properties of various learning algorithms. If the outputs are graded or analogue in nature then the accuracy required will depend upon the application itself. For digital applications, it would seem best to give a tolerance to the measure to avoid sampling noise. For example, if we take an output value of >0.5 to be 1.0 and <0.5 to be 0.0, then the slightest oscillation in the output about 0.5 will cause a lot of sampling noise, with the output flipping between logic true and false. The t-convergence measure suggested by Kolen and Pollack (1990) suffers from this drawback. Fahlman (1988) used criteria of <0.4 for 0.0 and >0.6 for 1.0.

A good method would seem to be to declare success when the sum squared error falls below a fixed threshold. This was the method originally proposed by Rumelhart, Hinton and Williams (1986a). For example, with any binary Boolean function we have four testable output states. If we use the criteria of <0.1 for zero and >0.9 for one, we are allowing a tolerance of 0.01 for each sum-squared error measurement. Over the four testable states, this gives an acceptable level of sum-squared error of 4*0.1*0.1 = 0.04. I use this calculation for determining whether or not an acceptable convergence had been achieved. It should be noted that using criteria of 0.1 and 0.9 as acceptable tolerances for digital outputs of 0.0 and 1.0 is different to setting the actual target values for the outputs to 0.1 and 0.9, which would give finite weight solutions.

The bifurcation mappings produced in the pilot experiments are simple scatter plots but despite their simplicity they do show a good global view of the eventual limiting error states that a given network will settle into. Unfortunately, if we are interested in convergence of the procedure to specific weight values, the fact that many different weight-space states could produce the same total sum squared error will exacerbate the crudeness of solely using these diagrams as a tool to view the dynamics. However, this factor becomes redundant where we are interested in the convergence of the procedure to specific error values, rather than to specific weight values.

Reporting of learning times becomes more complicated when there are problems with local minima. That is, if a few trials get stuck in a minimum where the error is high, then averaging the results of such trials with the results of totally successful trials produces a useless measure since this average has been polluted by poor data. It has become impossible to distinguish between methods with fewer failures or a better convergence average. Therefore, simple averaging of the results of learning can be meaningless for situations where local minima occur. This is one reason why we need a measure of *convergence* for benchmarking. In this chapter I shall focus on applying the concept of entropy to achieve this measure, in the sense that entropy can be used as an indication of the number of possible solutions to a problem.

One limitation of the bifurcation diagrams when used as the as sole statistic of the raw data is that the bifurcation plot lacks any indication of the relative frequencies of occurrence of the different possible eventual solutions. Additional statistics are required for this purpose. One candidate is a simple plot of the mean of all of the errors against the lambda parameter. Comparing both statistics together would give an indication of both the number of solutions, and the relative frequencies of those solutions. However, the mean of the error would still be susceptible to pollution from errors that are of relatively low frequency but of high absolute value.

A solution to this would be to develop a measure of convergence, or an entropy statistic. A measure of convergence would give an indication of the number of solutions to a given problem. When this measure has a value of zero, we could be sure that an associated mean of the error values would be true.

6.1 Entropy

The appearance of complex chaotic behaviour in bifurcation diagrams leads to the question of the relationship between chaos and statistical mechanics. One way to connect these phenomena is to apply the concept of entropy to a chaotic system, comparing the result to a statistical system.

We can start by considering a hypothetical statistical system for which the outcome of a certain measurement must be located within some interval. If the interval is subdivided into *N* subintervals, we can associate a probability *Pi* with the *ith* subinterval containing a particular range of possible outcomes. The entropy of such a system is then defined (Shannon, 1949) as:

$$H = -\sum_{i=1}^{N} p_{i.} \log p_i \qquad (11)$$

This quantity may be interpreted as a measure of the amount of disorder in the system or as the information necessary to specify the state of the system.

We can apply this formulation to a bifurcation diagram by establishing *N* 'bins' or subintervals of the unit interval into which values of an iterate may fall. If the outcome is known to be in a particular subinterval, then *H* will be zero, the minimum value.

In the nonchaotic state, the iterates will fall into relatively few of the bins, and entropy is low. In the chaotic state, the entropy is higher, and if the frequencies of occurrence are equal, then it approaches $log_n N$. The entropy approaches the maximum of $log_n N$, when the behaviour has become ergodic - by definition, an ergodic trajectory is one that is independent of previous conditions. *H* thus gives a measure of the solution set or number of solutions.

This formulation for H has some very useful properties, especially as a measure of the entropy of the limiting sum-squared error E of a system using the generalised delta rule. In particular, if we measure the sum squared error E of a network after a particular number of iterations and across a number of initial conditions, then when the entropy of E is zero, all trajectories will have converged to the same value of sum squared error.

If the behaviour under analysis has the statistical character of white noise then all the subintervals are equally probable so that $P_i = 1/N$ for all i. The entropy measure then reduces to $H=log_n N$, which can be shown to be its maximum value. Another way of saying this is that the amount of further information needed to specify the result of a measurement is at a maximum. Since we can always calculate this maximum value for $H=log_n N$ in advance, we can always normalise H, and so compute the normalised entropy of sum-square error for any network, sampled after a specific number of iterations. This constitutes a means of comparing the convergence properties of any generalised delta rule system.

A standard method for measuring entropy is to take only one initial condition per lambda value. In order to discard transients and get a measure of the long-term behaviour, the error state of the system can be sampled sequentially for a given number of iterations, after throwing away the error measurements from the first few iterations. This method has been used by many investigators, for example Renals (1990), Baker and Gollub (1990) and Bertels, Neuberg, Vassiliadis, and Pechanek (1995, 1998).

This standard method is sufficient when we are considering the simple one-dimensional logistic map but this is insufficient to characterise multi-layered delta rule systems of many dimensions where we wish to have a measure of entropy at an exact epoch and across many different initial conditions.

6.2 A Smoothed Entropy Statistic

In any real experiment, we would have to limit the number of training cycles to some acceptable figure, and so we might expect the actual measured error readings to converge to the limiting value plus or minus some very small fractional part. This small fractional part would reflect the different dynamics for all the different initial conditions, cumulative round-off errors, and so forth. Hung (1995) examines how to estimate the parameters of a chaotic system given noisy observations of the state behaviour of the system.

The residual error or measurement noise presents a problem for measuring the entropy of the limiting error for an artificial neural network, because we have only a limited number of trials from which to take measurements, and also we have to quantize the error measurement into a limited number of bins. This is done by subdividing the maximum range for the measured error into a certain number of bins and keeping a count in each bin of the number of values that fall into that subdivision.

If, for example, we have a system that converges to only one solution, to within a certain tolerance, we may find that the absolute value of this error is on the boundary of one bin and another. The tolerances of the error readings, however small, would be large enough to make the values fall into either bin with some certain probability, giving a spurious positive reading of entropy. We know that if the system similarly converged to an absolute value of error that had the same tolerance but was centred in the middle of a bin that we would end up with an entropy measure of zero.

This situation is typified in Figure 12, where we have some hypothetical noisy or fuzzy convergence data. The first grid overlaid upon the data, on the left hand side, represents an arrangement of bins where the absolute positioning of the bins is such that the data points fall into the top two bins. The second overlaid grid, on the right hand side, represents an absolute positioning of the bins such that the data points fall into only the middle bin. The point here is that the noisy data line is to the same tolerance in each case but the first grid will give a spurious positive reading of entropy, whereas the second grid will give a reading of exactly zero entropy.

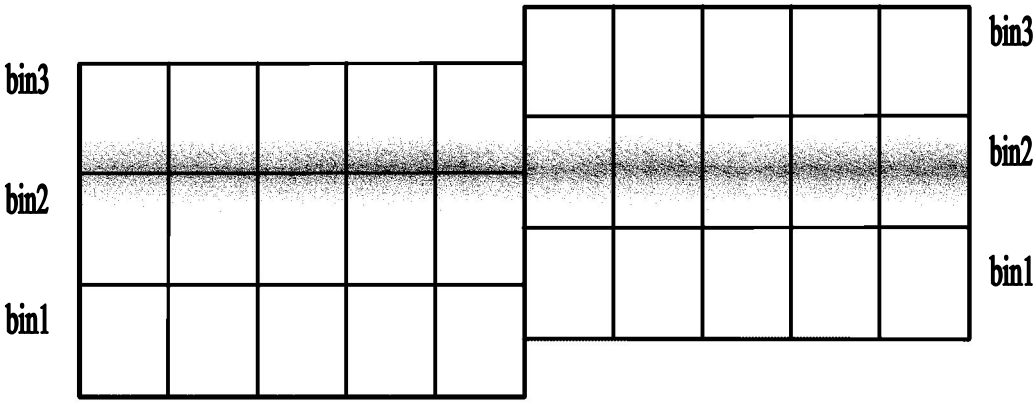


Figure 12. The measurement of entropy by dividing the sample space into discrete bins can be affected by the absolute location of the divisions.

We can however smooth out these spurious artefacts by making an average of a number of entropy readings obtained from shifting the absolute value of each bin by small increments from -0.5 to +0.5. The idea behind this is that the spurious data thrown up by the absolute value of the bin positions will be cancelled out across all transitions in this range. Table 3 shows an example code fragment in BASIC to illustrate the procedure.

Table 3. BASIC code fragment to illustrate calculation of smoothed normalised entropy

```
bins = 100                                      ‘ number of bins
passes = 100                                    ‘ number of passes
maxent = Log(bins)                              ‘ maximum possible entropy
H = 0                                           ‘ initialise H to zero

For s = 1 To passes
        Erase bin                               ‘ clear array
        offset = -0.5 + s / passes              ‘ offset goes from -0.5 to +0.5
        For n = 1 To bins
                index = 1 + Int(offset + bins * data(n))
                bin(index) = bin(index) + 1          ‘ increment bin counts
        Next n
        For n = 1 To bins
                p = bin(n) / bins
                If p > 0 Then H = H - p * Log(p)     ‘ accumulate H
        Next n
Next s
smoothed_normalised_entropy = H / (passes * maxent)
```

The code fragment given in Table 3 assumes we have an array of values we wish to calculate the entropy of, named **`data()`**, and a working array named **`bin()`** for intermediate calculations. Both arrays are of length **`bins`**. The algorithm works by shifting the absolute position of the bins with an offset, which is unique for each pass through the data. The final entropy value is an average of all of the passes. A reading of entropy that is smoothed using this method will tend towards zero if there is a definite positive frequency of errors that are all lower that the bin resolution, whatever the absolute position of the errors. The greater the numbers of averaging operations, the greater the smoothing effect. This algorithm is simple and effective, and a full working version is included in the program listings in Appendix B.

6.3 Application to the Standard Logistic Map

We can apply the smoothed entropy measure to the bifurcation mapping of the standard logistic map from Figure 5. This application to the standard logistic map is shown in Figure 13. This plot shows a comparison of the mean of the eventual values of x (black) with the eventual entropy of the values of x (grey), as a function of λ. The smoothing factor is 100.

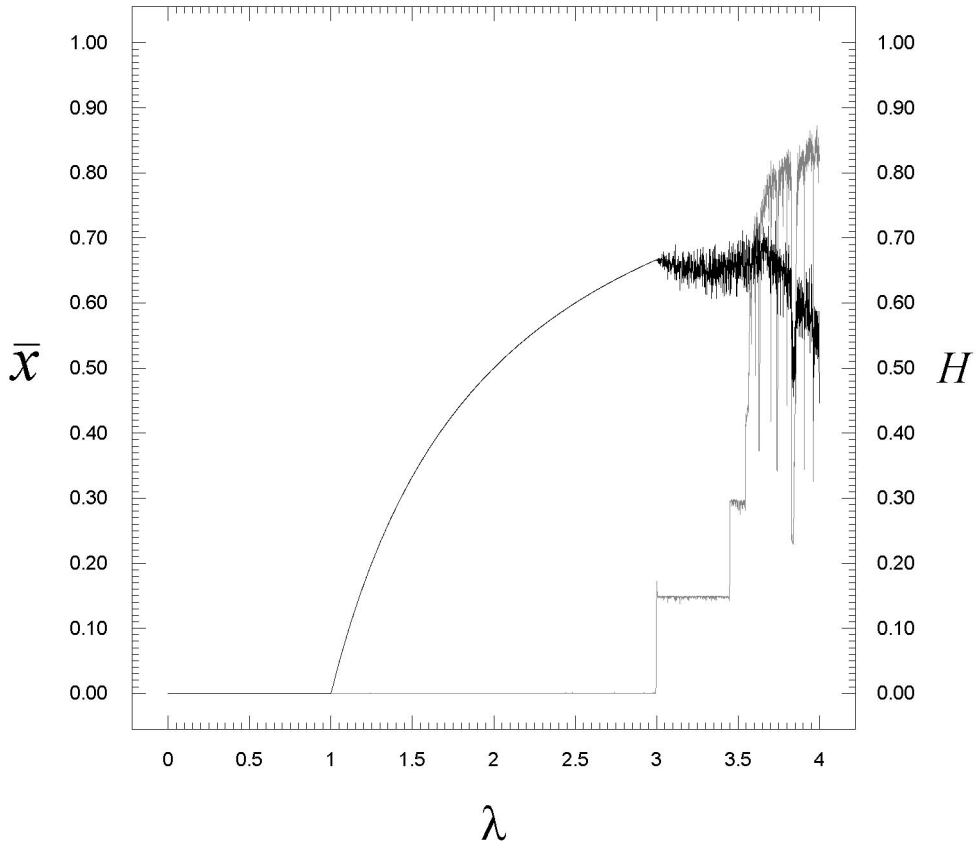


Figure 13. Comparisons of the mean (black) with the smoothed normalised entropy (grey), of the 1,000th iterate of the logistic function $\lambda x(1-x)$, as a function of the parameter λ. There are 100 initial values of x of between 0.1 and 0.9. The entropy measure is calculated using 100 sample bins and so normalised to $\log_n(100) = 1.0$. 100 smoothing passes are applied to the data. Overall, the maximum mean value of x is 0.726, occurring at $\lambda = 3.727$, and the maximum smoothed entropy value for x is 0.873, occurring at $\lambda = 3.983$.

The entropy versus mean error plots collapses the bifurcation diagram data from a many-to-many to a many-to-one mapping, simplifying the analysis. As expected, there is a generally an increase in entropy with an increase in λ, except in the downward spikes in the windows of periodic behaviour. The smoothed entropy measure *H* also discretely steps to higher values at each successive bifurcation point.

The downward spikes in the plot of entropy (grey) represent loss of information as to the initial conditions. The region or gap of lowest entropy therefore represents the most stable mode for the system across the initial conditions. The largest of these downward spikes occurs near $\lambda \approx 3.83$, where we know a stable 3-cycle occurs, a window of periodicity.

We can easily apply this method to measure the error convergence of a generalised delta rule system by comparing both the mean and entropy of sum-squared error. An entropy reading of zero when the mean error is low tells us that all the learning trials genuinely converge to a single solution with low error, and that the mean error does not contain spurious, low frequency occurrences of higher-error solution states.

This measure of entropy gives an indication as to information loss from a given range of initial conditions after a specific number of training iterations. The loss of information about the initial conditions as the iteration number increases arises from the noninvertability of the *O(1-O)* term, the logistic recurrence relation of Equation 9. That is, from Equation 9 it is possible to predict x_{n+1} from x_n but there is ambiguity in trying to retrodict x_n from x_{n+1}. One finds the same noninvertability with elementary functions such as *y=sin (x)*, because the inverses of folded functions can be defined only by limiting the original domains. This is why universality theory applies to non-linear functions that are folded, or have a quadratic maximum.

The loss of information about the initial conditions is also related to the ergodicity of the iterated sequence. Ergodicity can be defined as a property of a time-dependant process whereby the eventual (limiting) distribution of states of the system is independent of the initial state (Daintith and Nelson, 1995). Using our measure of entropy, a reading of zero tells us that all the learning trials converge to a single solution, from a range of initial conditions. Therefore, the convergence is shown as independent of that range of initial conditions. The measure also shows us points of maximum entropy, where the behaviour under analysis has become psuedo-random.

At some learning rate scaleable to the Feigenbaum number, output error values in a generalised delta rule system will become maximally ergodic. It follows that this will introduce chaotic (statistically random) error feedback into the system, of a character that is independent of both the actual transfer function of the system (as long as it is a folded function) and the initial conditions. It is possible that this type of feedback, if it satisfied the condition that it were greater in amplitude than the height of some or all of the local minima in the error function of the system, could tunnel directly to the global minimum. The fact that the route to the global error minima must be highly ergodic is advantageous because it coincides with the point of minimum dependence on initial conditions.

For a potential barrier of error in weight-space to be overcome requires that the system move from a point of relatively high error, and then through a point of even higher error before moving to a point of error lower than the first. A similar sequence is required to escape from a potential well. It is also possible that we may never “touch” the high region in between, but jump straight to the low region. The tunnelling through of regions where a potential barrier in error-space is crossed could be represented in the long term as a distortion of the error surface by a process of flattening the weight-space. In a sense this can be visualised as the system overcoming or being independent of bad initial conditions.

The high ergodicity of the system at or after the point of maximum entropy may overcome or potentially-tunnel through local minima. If this were the case this would show up as a sawtooth shape on a plot of entropy of error versus λ, leading up to the point of maximum entropy. The entropy would rise as the lambda parameter was increased, and when the behaviour was ergodic enough (and of the required amplitude) to overcome a local minimum of the error function, the entropy would be seen to fall, as more trajectories follow a universal route to the global error minimum. We are now in a position to be able to hypothesise that there could occur a kind of universal effect ergodically guaranteed to converge to the global minimum of sum squared error when increasing the λ parameter value past the point at which there is maximum entropy of error.

Chapter 7

Procedure

Having proved that there are presentation order-effects with the on-line method of updating the weights after each presentation, it was decided to dispense with the method of updating the weights after each presentation and instead choose the more accurate batch procedure of changing the weights after each epoch. This requires storing the weight increments over all presentations, and updating the weights with the summed increments computed from all pattern presentations after each epoch. The use of batch training should avoid input pattern presentation order effects and so simplify the analysis.

A series of programs were written to simulate a number of small feedforward networks using the above procedure. The programs were configured so that different initial weight ranges, number of epochs, and different ranges of the lambda parameter could be tried out. The networks were trained on the simple unary and binary Boolean functions as shown in Tables 1 and 2. All input patterns and output target values were set to precisely 1.0 or 0.0 for values of true or false, except for a special case where output targets were set to 0.9 and 0.1 so that finite weight solutions could be studied.

The topologies and functions to be tested were the bias-less 1:1:1 topology learning the identity function, the 2:2:1 fully biased, limited-feedforward topology learning the exclusive-or function, and, lastly, the 2:1:1 fully biased, full-feedforward topology learning all of the binary Boolean functions. In each experiment with each topology a different number of epochs was tried out. With each different problem, testing at a large number of epochs gives us a good idea of the long-term behaviour, and a low number of epochs gives us an indication of whether we can achieve convergence quickly.

Each program generated data files that could be analysed later. The data files used a standardised format for ease of subsequent analysis (the data file format and the program listings are given in Appendix B). Essentially, each file contained header information describing all the different parameter values ranges, and the rest of the file contained all the measured data points of the sum squared error of the system recorded for each different lambda parameter after the correct number of epochs. Each lambda parameter step value collected a given number of values of the sum-squared error.

A program was written to allow complete automation of the analysis of the data-files generated from the simulation runs. As well as producing the bifurcation diagrams, this program produced the mean sum squared error plots and the smoothed normalised entropy of error plots for the raw data. A useful feature of the program was the ability to calculate the lambda parameter values for the maximum and minimum of mean error and entropy of error.

A simple feature was built into the analysis program to compute the lambda parameter values for the first and last cases in the data file where all the errors were below a certain threshold of sum square error. I called this parameter the *SSE-level*. The reason for including this statistic is that if, in the limit, a discrete gap is found where all learning trajectories converge to below this level, then these first and last points will denote the starting and ending values of the parameter range for the gap.

The results given in the following chapters are from implementing the algorithms in Microsoft Visual Basic 6.0 on an Intel Pentium computer. Although not reported here, I have personally confirmed the optimal results in Visual C++ 6.0 and MASM 6.13 on the same computer, and in hand-coded assembler on a Motorola 6811 processor. The numbers for the optimal values and bifurcation points for the λ parameter turn out to be the same.

Chapter 8

Results and Analysis

This chapter presents the results from a number of experiments. Firstly, we consider the biasless 1:1:1 topology learning the identity function, and a graphical analysis of the convergence mechanism is given. Next, the problem of the 2:2:1 fully biased, limited-feedforward topology learning the exclusive-or function is examined, using output target values of precisely 1.0 or 0.0 for true or false, and additionally using output target values of precisely 0.9 and 0.1 so as to study finite weight solutions. Finally the 2:1:1 fully biased, full-feedforward topology learning all of the binary Boolean functions (given in Table 2) is considered.

8.1 Benchmarks for the 1:1:1 net Learning Identity

It was decided to map the dynamics of the 1:1:1 identity problem out to very high lambda values in order to see if there were any dynamical effects that were not detected in the pilot study. We recall that in the pilot study, when a 1:1:1 network was tested with the identity function, relatively large initial values for the weights were used, with $-10 < w_x < +10$. A few test runs using very high lambda values (up to $\lambda = 200$) show that dynamical effects do indeed occur at higher parameter values. In this section I will show evidence for these effects, which include bifurcation phenomena and a region in the range of the λ parameter where convergence to the global error minimum can be ensured.

It was noticed from some test runs that the dynamical effect of global convergence is readily apparent when we use a relatively *low* range for the initial weight values. The same effect can be shown when using large initial weight values, but the effect can require many more training iterations to become apparent. Since one of the aims of this study is to ascertain whether it is possible to achieve global convergence using the lowest possible number of training iterations, a low initial weight range was therefore selected.

An optimal weight range can be determined empirically, by producing a number of bifurcation diagrams and an associated mean sum squared error/entropy of error map for the problem in hand, in each case progressively lowering the initial weight range. At the higher initial weight value ranges, we see the maximum number of solutions, representing the maximum number of error minima, and also we have the possibly that the units may well be saturated, leading to very slow gradient descent. The experiments in the pilot study yielded results of this nature.

As the initial weight range is progressively lowered we eventually find an optimal range, where we find the minimum number of local error minima. It was found by this method that the optimal range for the initial weight values for the 1:1:1 network when trained on the identity function is $-0.1 < w_x < +0.1$.

Three simulation runs were performed with this range of initial conditions, testing the problem to 100, 250, and 500 epochs, using a learning rate parameter range of $0 < \lambda < 200$. The bifurcation maps for these epoch-numbered simulation runs are given in Figures 14, 16 and 18, respectively.

The results of the subsequent analysis of the simulation runs are summarised in Table 4. Note that the value of maximum entropy and the lambda parameter at which it occurs remains constant across all training epochs. The lambda parameter value corresponding to the point of minimum entropy gradually approaches this constant value as the epoch number is increased. This factor coincides with the lambda parameter value for the start of the region where there is a clearing of points in the sequence of bifurcation diagrams. At the lowest epoch numbers, this clearing is quite fuzzy, showing rough convergence to the global minimum of the error function (Figures 14 and 15) along a parameter range of $100 < \lambda < 140$. As the epoch number is increased, this region becomes completely clear, becoming a universally convergent gap along a parameter range of about $100 < \lambda < 180$ at 500 epochs (Figures 18 and 19).

Table 4. Results for training a 1:1:1 network without biases on the identity function. Learning rate parameter range is $0 < \lambda < 200$ and initial weight condition range is $-0.1 < w_x < +0.1$.

EPOCH	MAXIMUM ENTROPY		MINIMUM ENTROPY		MINIMUM SSE		CONVERGENT GAPS (*SSE*<0.27)	
	H	λ	*H*	λ	*SSE*	λ	λ_{FIRST}	λ_{LAST}
100	0.54	98.8	0.00	127.2	0.25	141.4	111.5	147.2
250	0.54	98.8	0.00	122.9	0.25	161.8	102.6	166.7
500	0.54	98.8	0.00	102.6	0.25	170.3	101.6	178.6

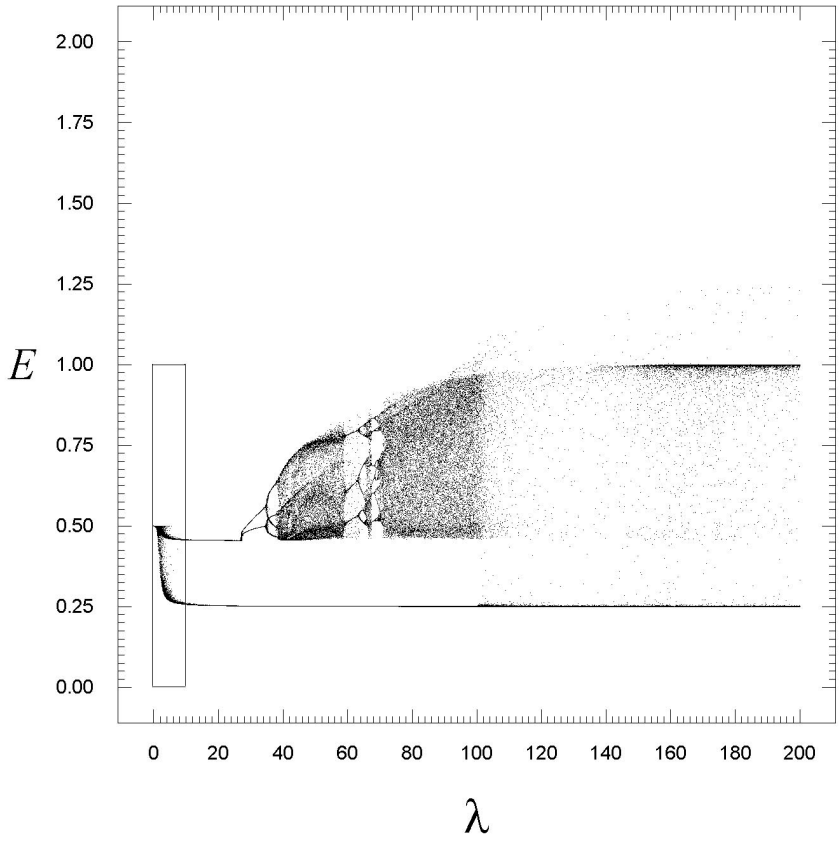


Figure 14. The sum squared error for the 1:1:1 unbiased network trained on the identity function and sampled at 100 epochs plotted against λ. Sum squared error is in the range [$0 < E < 2$] and λ is varied over the range [$0 < \lambda < 200$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 20,000,000 training epochs. The first convergent gap where all *S.S.E.* <= 0.27 occurs at $\lambda = 111.5$ and the last at $\lambda = 147.2$.

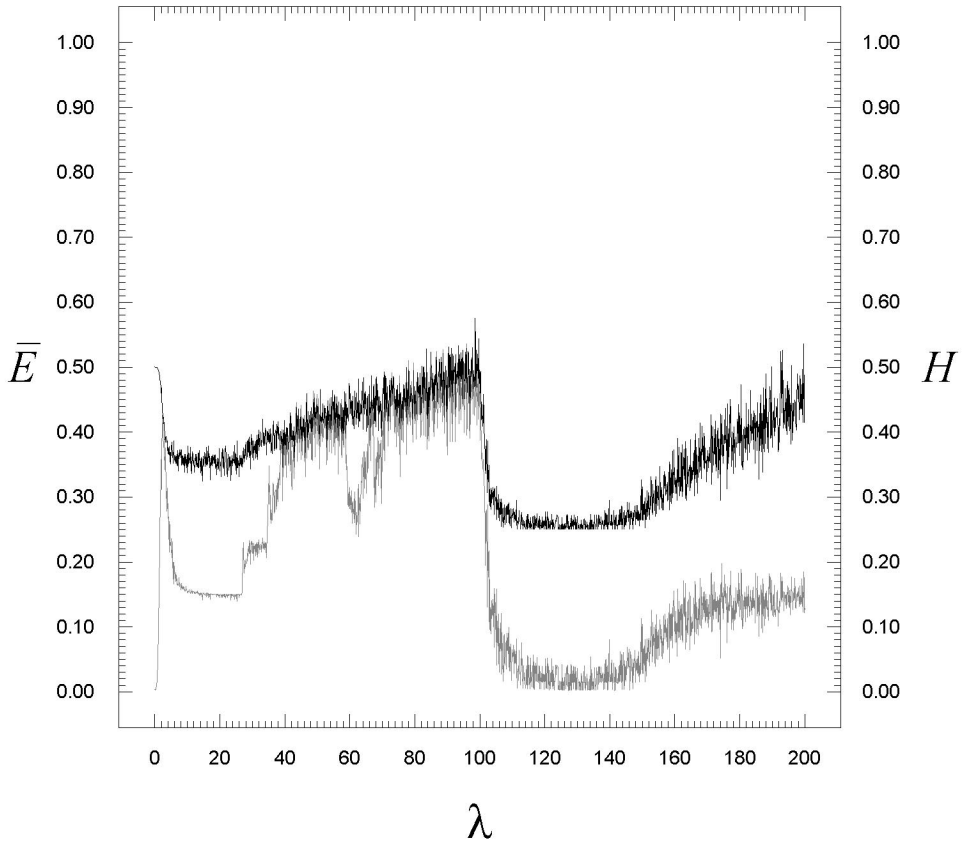


Figure 15. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 13. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 1]$. The maximum of E is 0.576, at $\lambda = 98.8$. The maximum of H is 0.543, at $\lambda = 98.8$. The minimum of E is 0.250, at $\lambda = 141.4$.

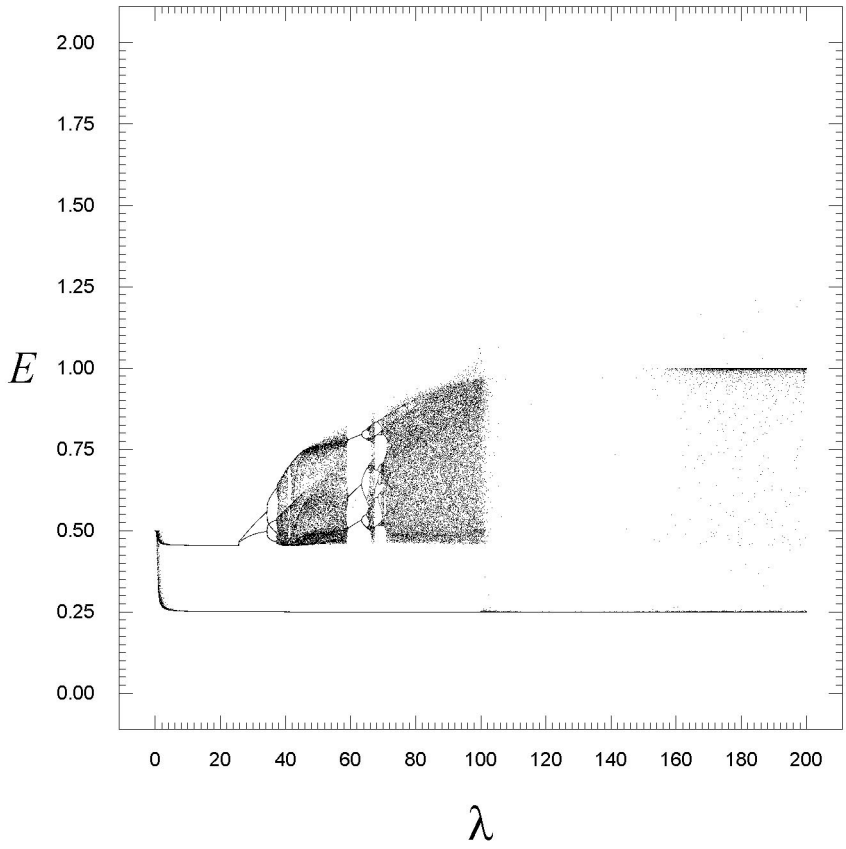


Figure 16. The sum squared error for the 1:1:1 unbiased network trained on the identity function and sampled at 250 epochs plotted against λ. Sum squared error is in the range [$0 < E < 2$] and λ is varied over the range [$0 < \lambda < 200$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 50,000,000 training epochs. The first convergent gap where all *S.S.E.* <= 0.27 occurs at $\lambda = 102.6$ and the last at $\lambda = 166.7$.

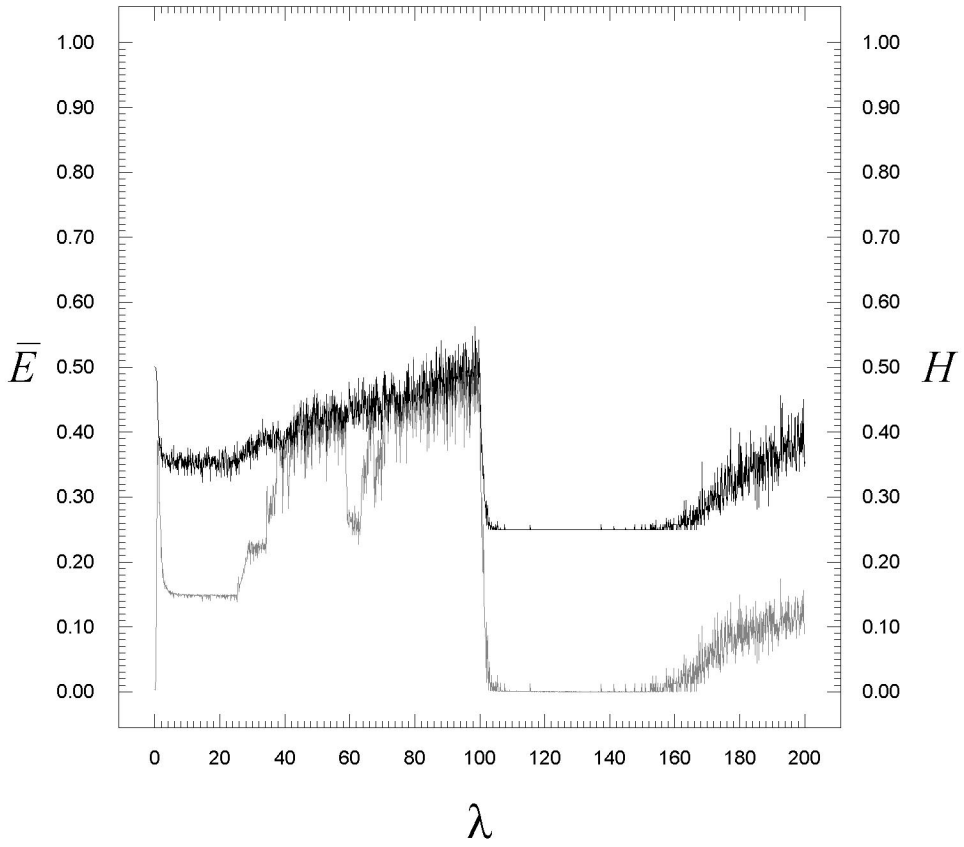


Figure 17. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 15. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 1]$. The maximum of E is 0.563, at $\lambda = 98.8$. The maximum of H is 0.538, at $\lambda = 98.8$. The minimum of E is 0.250, at $\lambda = 161.8$.

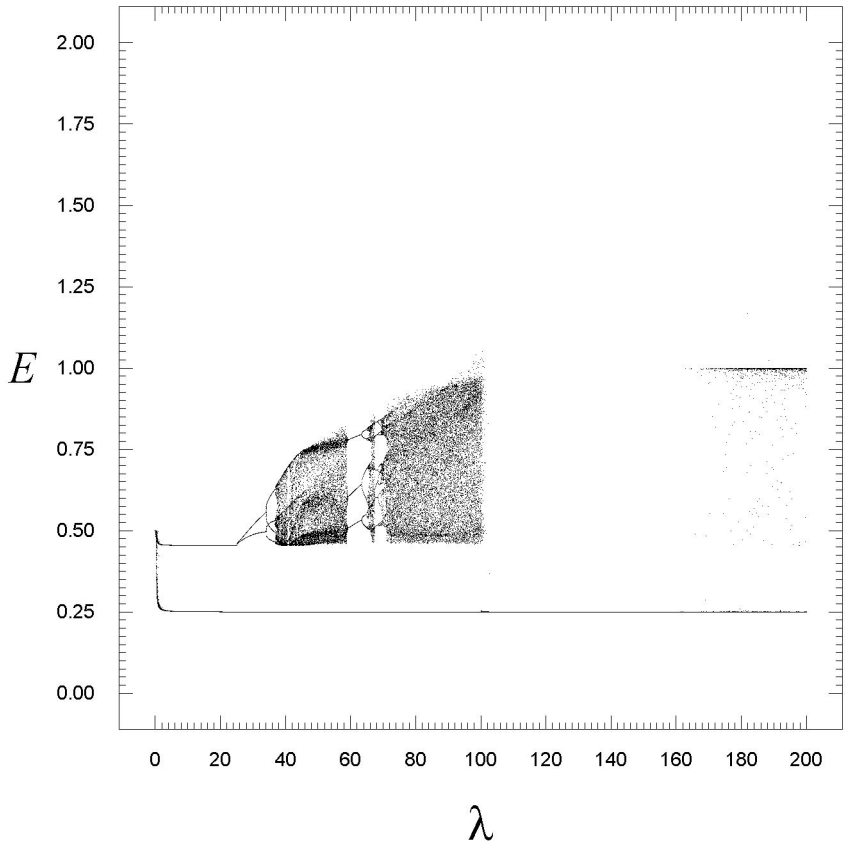


Figure 18. The sum squared error for the 1:1:1 unbiased network trained on the identity function and sampled at 500 epochs plotted against λ. Sum squared error is in the range [$0 < E < 2$] and λ is varied over the range [$0 < \lambda < 200$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 100,000,000 training epochs. The first convergent gap where all S.S.E. <= 0.27 occurs at $\lambda = 101.6$ and the last at $\lambda = 178.6$.

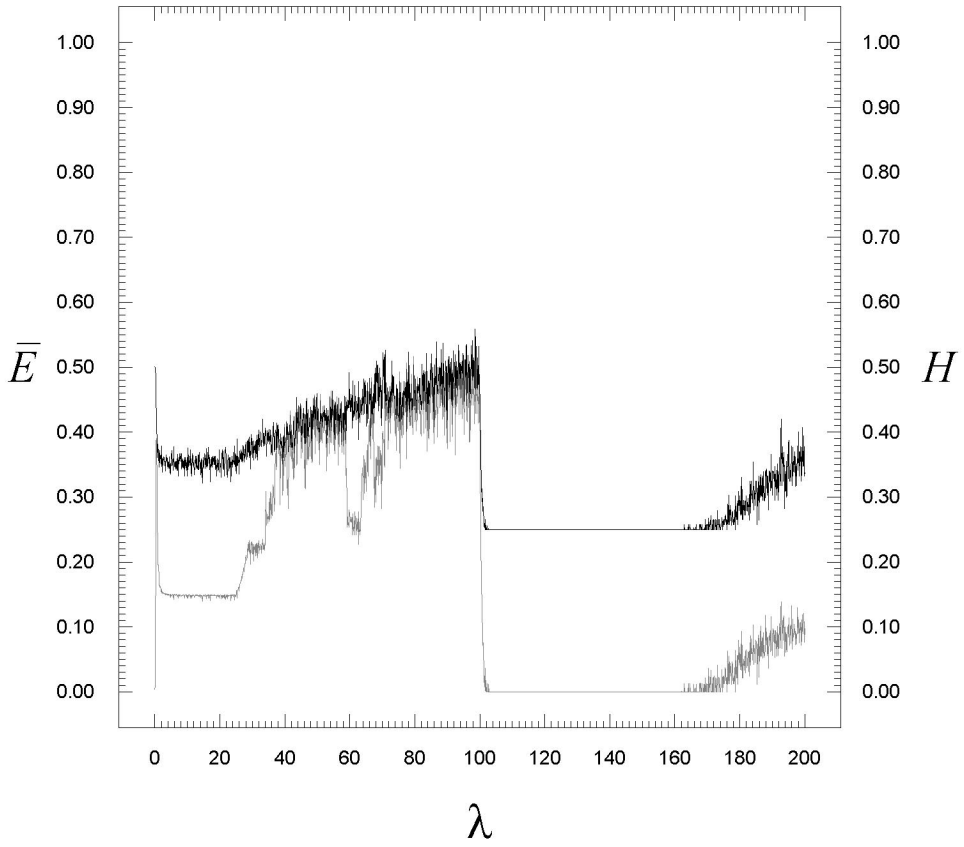


Figure 19. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 17. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 1]$. The maximum of E is 0.559, at $\lambda = 98.8$. The maximum of H is 0.542, at $\lambda = 98.8$. The minimum of E is 0.250, at $\lambda = 170.3$.

Figure 14 shows the first bifurcation mapping, taken at 100 epochs. This diagram has a small rectangle overlaid to show the region of the dynamics from $0 < \lambda < 10$ measured in the pilot study (Figure 7). We can see from the plot that at just past $\lambda = 25$, the solution-line representing the local minimum of error will undergo a bifurcation sequence. This sequence cascades until we reach $\lambda = 100$, at which point the line representing convergence to the local minimum begins to break up and the points move to the global minimum.

Clearly visible in the mean error versus entropy diagram of Figure 15 is the phenomenon of the entropy reaching a peak value before the system converges globally, which suggests that the convergence to the global minimum of error is related to the system reaching a condition of maximum entropy. This transition occurs at a value very close to $\lambda = 100$, reaching a minimum along the range $120 < \lambda < 140$. After this, the values of both measurements begin to rise again.

Some areas within the chaotic region are periodic. The downward spikes show these areas in the associated entropy mapping in Figure 15. The region where $\lambda = 60$, for example, forces the system into a 3-cycle, and has the largest downward spike in the chaotic region of the diagram. It is therefore the most stable area within the chaotic region.

Looking at the series of bifurcation mappings produced from these experiments (Figs. 14,16,18) together with the associated mean error versus entropy diagrams (Figs. 15,17,19) one can see that a gap begins to form in the long-term error for the problem along a specific range of the λ parameter. This gap, occurring roughly between $100 < \lambda < 180$, is clearly one where points of error that are stuck in the local minimum for the problem undergo successive period-doubling bifurcations, until the behaviour of the points become ergodic enough to escape the local minimum and move to the global minimum. The underlying mechanism for this is probably due to the expansion of the attractor representing the local minimum until it crosses the separatrix (see Figure 6) and forms a route to the global minimum, thereby allowing these globally convergent trajectories.

We know that it is a simple matter for the 1:1:1 biasless network to solve the other unary Boolean functions of TRUE and FALSE because there exist no local minima for these problems. Although not shown here, the dynamics of the 1:1:1 biasless network learning the unary NOT function displays exactly the same property of universally converging to the global error minimum as for the IDENTITY function, and for exactly the same range of the lambda parameter. Armed with this knowledge we can say that this parameter region guarantees convergence to the global minimum of sum-squared error for a 1:1:1 biasless network solving all the Boolean unary functions.

The absolute minimum value of sum-squared error occurs just before the end of the gap of global convergence, and the absolute minimum value of entropy occurs after the start of the gap. The fact that the lambda parameter value at the point of maximum entropy is coincident with the start of the convergence gap, and is to a degree independent of the epoch number, means that it is an excellent predictive indicator for this type of universal global convergence behaviour.

The 1:1:1 identity problem without biases will require a different number of iterations to converge for each different initial weight-space vector. In the universal globally convergent region, the problem requires a minimum number of iterations to jump from the local to the global minimum, probably due to the highly ergodic error convergence and the shape of the error surface itself. The dynamic route to convergence at high lambda parameter values is, therefore, not simply an effect of varying the sampling distance. It is a phenomenon that is universal, in that it is characterised by successive period doubling behaviour as determined by the number F, and ergodic, in the sense that the solution has become partly independent of the initial conditions.

8.2 Graphical Analysis of Convergence Mechanism

We are in the fortuitous position of knowing, from the mapping results of the previous section, that there exists a discrete gap of universal global error convergence for the biasless 1:1:1 network when trained on the identity function. We know that this gap occurs when the lambda parameter value is varied between 100 and 180. The effect should guarantee convergence to the global minimum of sum-squared error in the long-term after at most a few hundred epochs.

Because there are only two weights for this network we can plot a contour map of the error surface, the same as the plot from Figure 6, and overlay actual learning trajectories at different lambda parameters and different initial conditions, in order to directly see the mechanism behind the convergence effect.

Figures 20, 21, and 22 show what happens when we run the simulation for 1,000 epochs using three different lambda parameter values. For each figure, three cases of initial conditions are chosen. The figures show the resulting learning trajectories taken for each of these cases. The three different starting conditions are:

Case 1: $w_1 = 2$, $w_2 = -6$ starts within domain of global minimum, right hand side

Case 2: $w_1 = -4$, $w_2 = 4$ starts within domain of global minimum, left hand side

Case 3: $w_1 = 4$, $w_2 = 4$ starts within domain of local minimum

Each successive epoch produces a new weight configuration, which is plotted as a small circle, with lines connecting successive configurations. The starting and ending configuration for each condition is labelled with the case number, 1, 2 or 3.

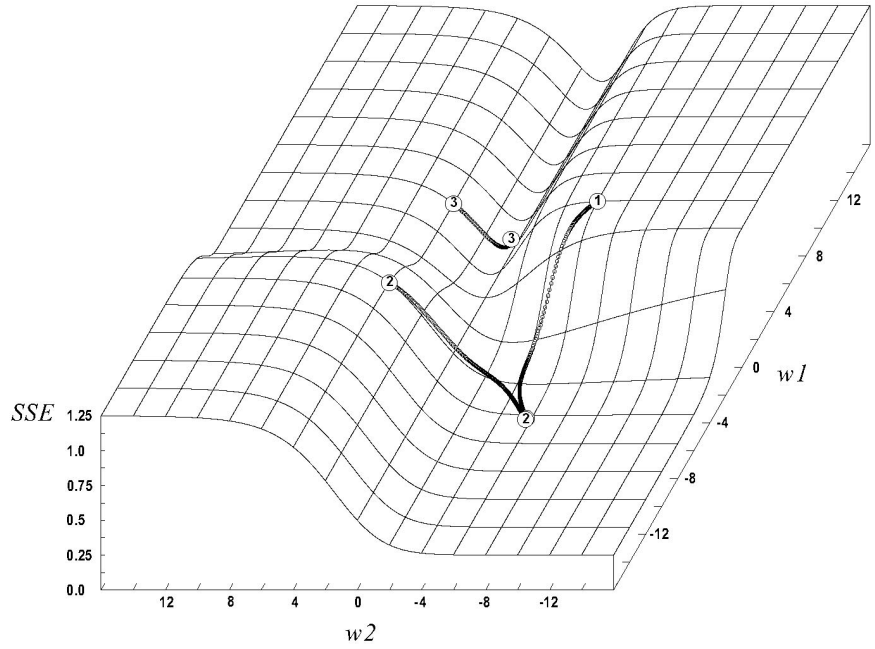


Figure 20. A 3-D contour map of the error surface of the 1:1:1 network learning the identity function. Learning trajectories over 1,000 epochs for the three initial condition cases are shown, all with $\lambda = 1$. In this isometric plot, the horizontal axis represents w_2 from +16 to -16. The other axis represents w_1 going from -16 at the front to +16 at the back. The height represents the sum-squared error for the weight configuration.

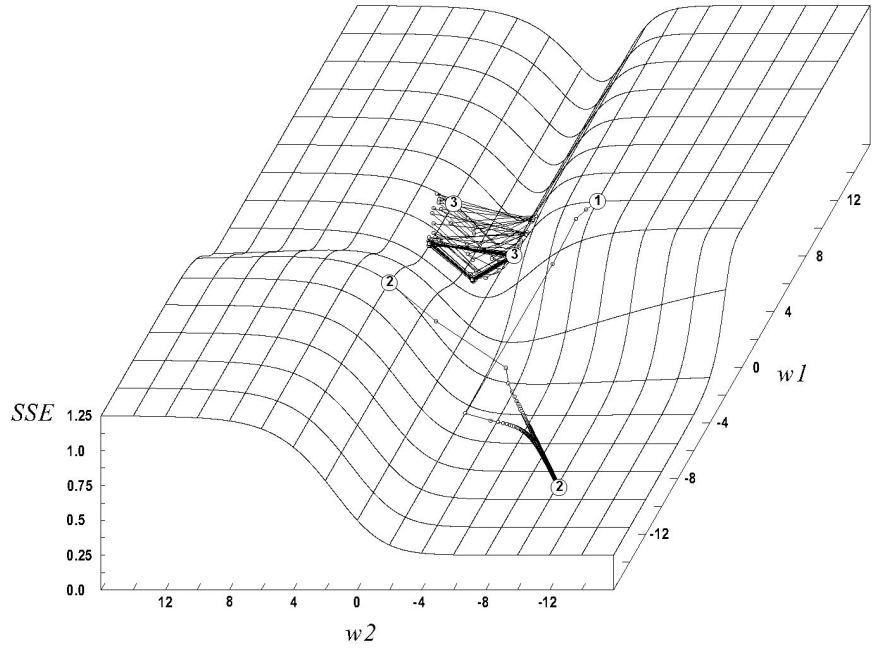


Figure 21. A 3-D contour map of the error surface of the 1:1:1 network learning the identity function. Learning trajectories over 1,000 epochs for the three initial condition cases are shown, all with $\lambda = 60$. In this isometric plot, the horizontal axis represents w_2 from +16 to -16. The other axis represents w_1 going from -16 at the front to +16 at the back. The height represents the sum-squared error for the weight configuration.

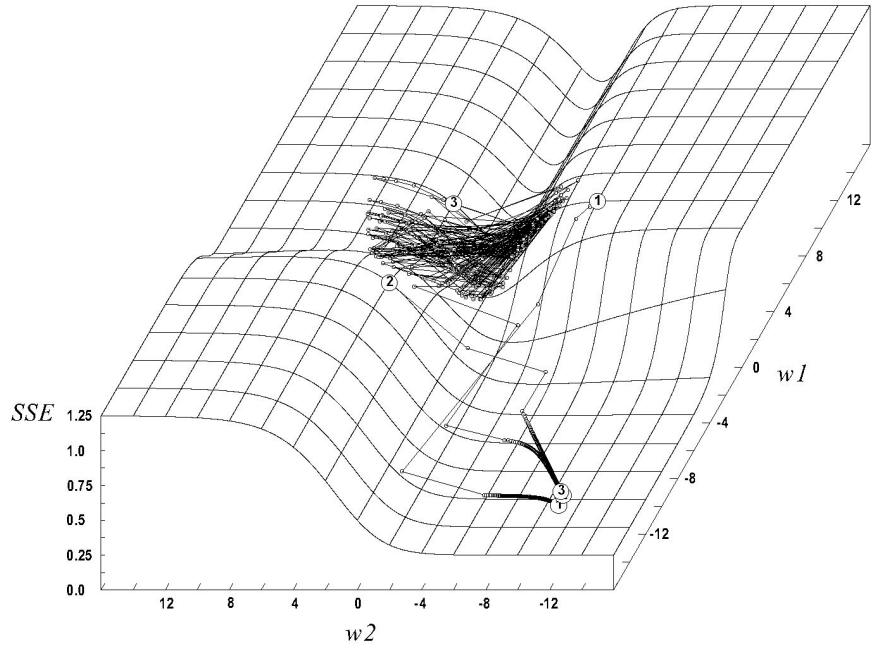


Figure 22. A 3-D contour map of the error surface of the 1:1:1 network learning the identity function. Learning trajectories over 1,000 epochs for the three initial condition cases are shown, all with $\lambda = 101$. In this isometric plot, the horizontal axis represents w_2 from +16 to -16. The other axis represents w_1 going from -16 at the front to +16 at the back. The height represents the sum-squared error for the configuration.

These error-surface plots with example learning trajectories do clearly show the mechanism behind how the universal global solution is achieved. The first diagram (Figure 20) shows what kind of trajectories we get when using low lambda parameter values. At the low parameter values, the system does gradient descent using small steps, and can get caught in the local minimum if we have “unlucky” initial conditions, such as the trajectory of case 3 in Figure 20.

The second diagram (Figure 21) shows the behaviour of the same initial points as the system is run with a lambda-parameter value of 60. From the bifurcation diagram of the same problem, we can see that at this parameter value the system oscillates in a 3-cycle when a point is in the vicinity of the local minimum. One can see that when the system is in the local minimum, it oscillates upon the error surface with an amplitude that is not enough to escape the minimum. The point remains in the domain of the local minimum. As the lambda parameter was increased, the frequency of oscillation of the trapped point was found to increase to a 6-cycle, at a lambda parameter value of 64. This behaviour is in agreement with the bifurcation maps for the problem, shown in Figure 18.

One can also see that the general direction of the learning trajectory for the point started in the local minimum (case 3) is upward, in the sense that successive points are moving up the error gradient, towards the separatrix (compare the trajectory for the same case with the previous figure) – the point in the local minima, at this parameter value, will undergo gradient ascent, instead of descent – i.e., we are slowly climbing up out of the minimum.

Figure 22 shows what happens when we run the system using a lambda parameter of 101, which is just past the point of maximum entropy, at the start of the gap of global convergence. All three initial points converge to the global minimum of the error surface. The point that was started in the minimum has oscillated against the walls of the minimum with just enough amplitude to escape. The point has lost its reliance upon the initial conditions, and so the trajectory of the point has become, to a degree, ergodic.

The underlying mechanism that enables the globally convergent trajectory of case 3 is probably due to the expansion of the attractor representing the local minimum, caused by the increase in the value of the λ parameter. For the λ parameter value of 101 shown in Figure 22, the expansion is such that the attractor has just crossed the separatrix, forming a connection or route to the global minimum. Once the point has crossed the separatrix, it is within the domain of the global error minimum. There is a corresponding large decrease in the error, and an increase in stability. This leads to a decrease in the likelihood of oscillation, and the point normally converges to the global minimum. This is the remarkable mechanism of universal global convergence.

This mechanism explains why the gap showing complete convergence in the bifurcation diagrams of Figure 18 is a "true" gap, and does not reflect, for example, a two-cycle of which we only see the "even" value, which happens to be a low-error value. Even if a two-cycle were possible here, once in the domain of the global minimum, such a point would normally converge just the same as the other two cases, because the large decrease in the error leads to an increase in stability inside this region.

The trajectory of case 3 in Figure 22 is certainly not a simple one. Here we are reminded of the suggestion of Kolen and Pollack (1990) that we might think of a many-body or multiple-attractor metaphor for convergence, instead of strict local gradient descent, with its theoretical reliance on an infinitesimally small lambda parameter. At the optimal value for the learning rate parameter, the trajectory is chaotic, and the basin structure breaks up, with the previously separate attractors corresponding to the two different sets of initial conditions joining together to form a single attractor.

We have shown that very low values for the learning rate parameter can only, at best, converge to the nearest minimum of the error surface from the given starting conditions. When using very low values for the learning rate parameter we are relying, to some degree, on the initial conditions being "lucky" in that they will eventually lead to an acceptable solution. On the other hand, at a relatively high optimal value for the lambda parameter, the global minimum of the error function has become a global attractor for the system, ensuring that long-term behaviour is independent of the initial conditions. We have therefore shown empirically that the best way of avoiding local minima, at least in this example, is to use very high values for the lambda parameter, rather than very low values, as originally suggested by Rumelhart, Hinton and Williams (1986a).

8.3 Benchmarks for the 2:2:1 net Learning Exclusive-or

The problem of training the 2:2:1 limited topology on the exclusive-or function gives an example of a dynamical situation known as degeneracy. Degeneracy occurs when a dynamical system cannot settle into a single solution but instead settles into one of two possible solutions. In the strictest sense, degeneracy occurs when the frequency of occurrence of both solution modes is equal (Daintith and Nelson, 1995). The 2:2:1 limited topology trained on the exclusive-or function is one of the hardest problems to solve using the generalized delta rule because the system can settle into a number of non-optimal solutions.

The initial range of weight values for the problem was again calculated by the method of using progressively smaller initial conditions until we arrive at a bifurcation map showing the minimum number of solutions. This gives an initial condition range of $-0.1 < w_x < +0.1$. A larger range of initial conditions will show more solution lines representing additional error minima, up to three minima for a range of $-10 < w_x < +10$, as was shown in the pilot study.

Two simulation runs were performed, testing the dynamics of the problem at 250 and 1,000 epochs in the range $0 < \lambda < 50$. The bifurcation maps for these tests are given in Figure 23 and Figure 25 respectively. Figure 24 and Figure 26 show the mean sum squared error and smoothed normalized entropy plots for the data. Summary results are given in Table 5.

We can see from Figure 23 that at the lowest values of the λ parameter we have a single solution line of *S.S.E.* = 1.0. This is most probably due to the network being in the "undecided" state where all output values are 0.5, the network has not had enough iterations to separate the input patterns. This situation is sometimes referred to as a temporary minimum (Ampazis, Perantonis and Taylor, 1999; Liang, 1991). Temporary minima are characterized by the equality of outputs corresponding to all training patterns (Lisboa and Perantonis, 1991). A temporary minimum can be recognized in the sum square error versus epoch curve, as an approximately flat part in which the sum squared error is initially virtually constant for a long training time. After a generally large number of epochs, this part in the energy landscape is abandoned, resulting in a significant and sudden drop in the sum squared error (Woods, 1988; Murray, 1991).

As the λ parameter is increased we see that the problem is degenerate, in that there are no points along the range of λ where one single solution forms. Instead, the best case we have is where two solutions form, one a local minimum of about *S.S.E.* = 0.66 and the other the global minimum where *S.S.E.* = 0.0, both occurring at $\lambda \approx 6$. This dynamical mode lasts until around $\lambda = 10$, at which point the global solution disappears, and only convergence to the local minimum remains possible.

As the parameter is increased past $\lambda = 10$ we see a now-familiar bifurcation cascade which continues until a λ value of around 25, where we see that the period-doubling sequence becomes ergodic enough to escape this minima. This transition coincides with a maximum of the entropy plot for the problem at $\lambda = 25$, shown in Figure 24. However, we can see that this universal-mode solution can move not only to the global minimum but also up to a solution representing a sum squared error of 2.0, which is an even worse local minimum that the first.

From the bifurcation diagram of Figure 23, we cannot tell the relative frequencies of the solutions, and there is no clear discrete gap allowing convergence to the global minimum. However, by comparing the plots of the mean error to the entropy of the error in Figure 24 we can still calculate the optimal system parameters. For the case of the first dynamical mode, we see a large downward spike in the value of the mean sum squared error at $\lambda = 6.3$, which is preceded by a large upward spike in the entropy of the error. This point is the first maximum of H and occurs at $\lambda = 6.0$, signifying incomplete convergence at the start of the optimal region.

Table 5. Results for training a 2:2:1 limited feedforward network on exclusive-or. Learning rate parameter range is $0 < \lambda < 50$ and initial weight condition range is $-0.1 < w_x < +0.1$.

EPOCH	MAXIMUM ENTROPY		MINIMUM MEAN *S.S.E.*		
	H	λ	MEAN	CONVERGENCE TO *S.S.E* < 0.04	λ
250	0.58	6.0	0.05	86/100	6.3
1,000	0.22	5.5	0.01	99/100	6.0

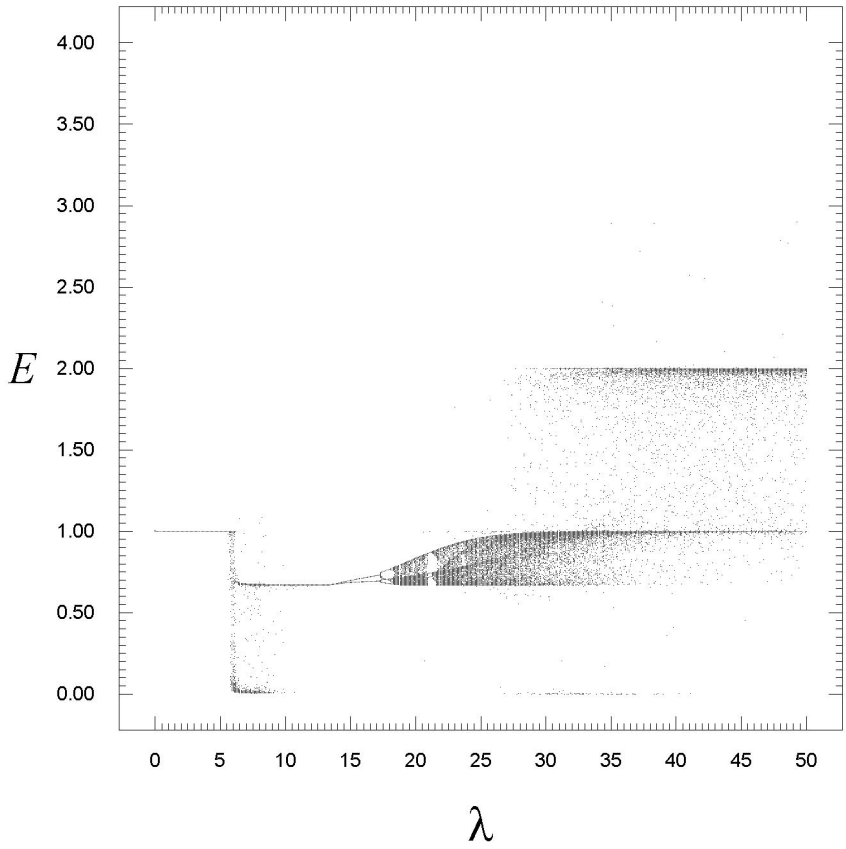


Figure 23. The sum squared error for the 2:2:1 biased, limited-feedforward network trained on the exclusive-or function and sampled at 250 epochs, plotted against λ. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 50$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 12,500,000 training epochs. No convergent gaps where $E <= 0.04$ were found.

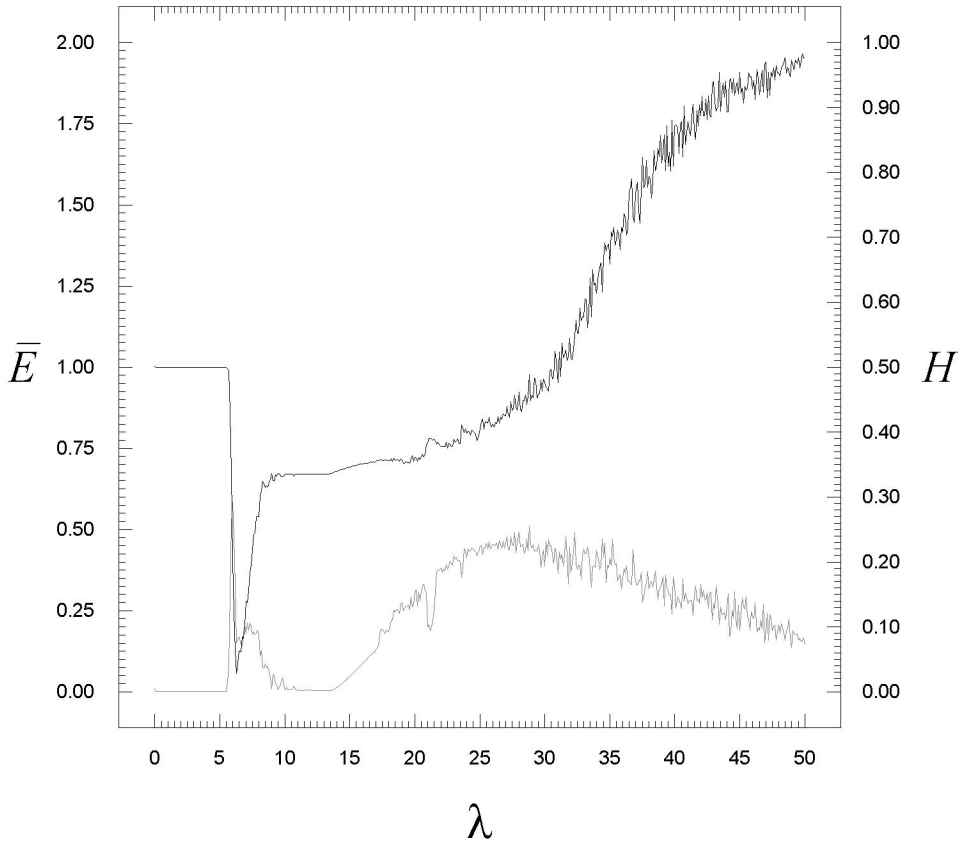


Figure 24. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 22. Entropy is calculated using 100 sample bins and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of E is 1.966, at $\lambda = 49.8$.The maximum of H is 0.585, at $\lambda = 6.0$. The minimum of E is 0.056, at $\lambda = 6.3$.

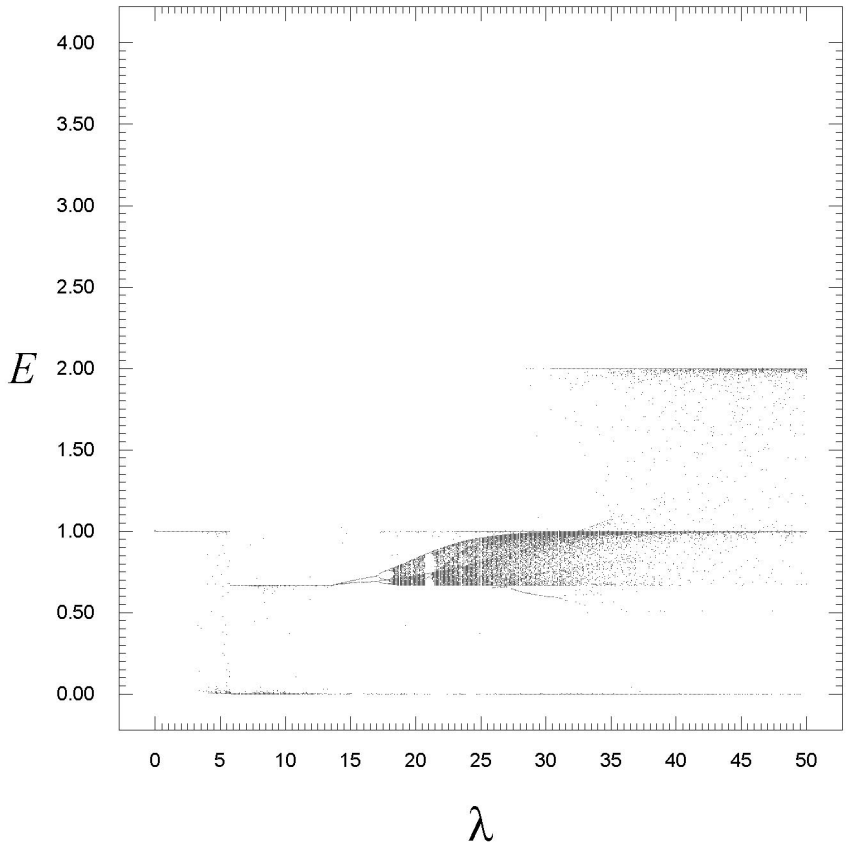


Figure 25. The sum squared error for the 2:2:1 biased, limited-feedforward network trained on the exclusive-or function and sampled at 1,000 epochs, plotted against λ. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 50$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 50,000,000 training epochs. No convergent gaps where $E <= 0.04$ were found.

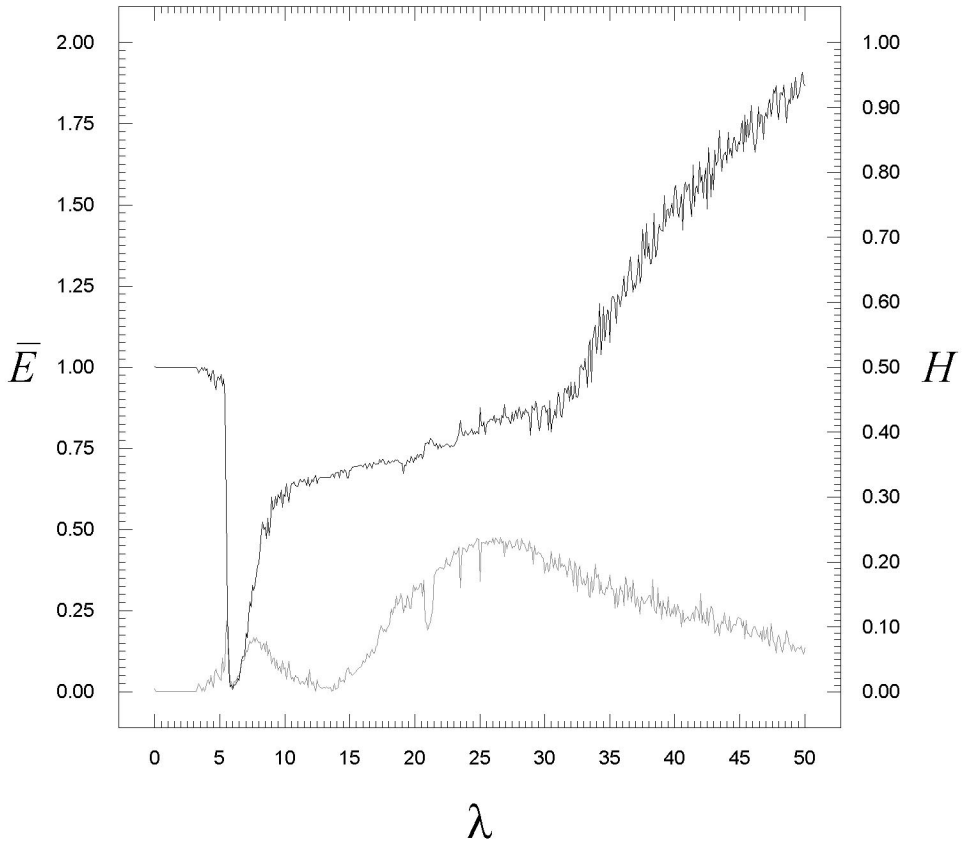


Figure 26. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 24. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of E is 1.909, at $\lambda = 49.8$.The maximum of H is 0.220, at $\lambda = 26.6$. The minimum of E is 0.008, at $\lambda = 6.0$.

Although there is a large downward spike in the mean error, this does not represent a discrete convergent gap. At 250 epochs the value of the smoothed normalised entropy at the lambda parameter corresponding to the minimum of E gives $H = 0.15$, indicating that many initial conditions will converge to the local minimum in this region.

The dynamics of the system at the transition coincident with the maximum entropy of error at $\lambda = 25$ can be analysed similarly. From Figure 24 we can see that the mean sum squared error slowly rises to a maximum value of 2.0 after this point, whilst the entropy slowly falls. This suggests that the system will converge to a local minimum of *S.S.E.* = 2.0 in the long term.

A system is degenerate in the strictest sense when the frequencies of occurrence of its modes of operation are exactly equal. Taking a rather loose use of this concept, we see that the results for the 2:2:1 problem at the optimal value of around $\lambda = 6$ shows degeneracy, in that two solutions form, one a local minimum and the other a global minimum. At a certain low number of epochs, the frequency of occurrence of these solutions will indeed be equal, but with increasing epoch numbers the system will converge to a global as opposed to a local minimum.

The higher likelihood of the frequencies of global convergence at higher epoch numbers suggests that the generalised delta rule can therefore bring the system out of degeneracy, in the long-term. However, there is still no discrete gap in a bifurcation map of this problem, and therefore we cannot prove *guaranteed* global error minimum convergence across all initial conditions.

The benchmarking procedure supplies the optimal lambda parameter where we have best odds of getting to the global minimum. The point of optimal convergence happens just past the point of maximum entropy. In the optimal region, the actual eventual errors are increasingly toward the global minimum with increasing number of training epochs.

8.4 A 2:2:1 Application with Finite Weights

The logistic activation function used with the generalised delta rule cannot achieve the extreme values of 1.0 or 0.0 without an infinitely large net input to a unit. Since this thesis is primarily concerned with the convergence of the generalised delta rule to an acceptable *error* value, and an acceptable error value can in general be reached many iterations *before* the weights must take on infinite values, the examples used throughout the thesis have used the idealised system whereby we use target values of 1.0 and 0.0 to represent Boolean true and false values. Although uncommon, this idealised system has been studied by others, for example see Bertels, Neuberg, Vassiliadis, and Pechanek (1995).

Most authors (for example Rumelhart, Hinton and Williams, 1986a; Lisboa and Perantonis, 1991; Sprinkhuizen-Kuyper and Boers, 1996b) have studied the 2:2:1 exclusive-or problem from the viewpoint of being specifically interested in the *weight* values. These studies utilise target values of 0.9 and 0.1 to represent Boolean true and false, since these target values ensure convergence to solutions with finite values for the weights from hidden units to output units. Additionally, when all weights converge to finite values, all patterns are learned exactly and the error is exactly zero.

Here I will apply the benchmarking procedure to the 2:2:1 exclusive-or problem using target values of 0.9 and 0.1, giving finite weight solutions. I will then show how the optimal parameters given by the benchmarking procedure can be applied to a number of known local error minimum conditions with finite weights reported in the literature.

Two simulation runs were performed, testing the problem at 1,000 and 25,000 epochs in the range $0 < \lambda < 50$. The bifurcation maps for these tests are given in Figure 27 and Figure 29 respectively. The same method of progressively lowering the range of initial conditions until a range that leads to the minimum number of solution lines on a bifurcation diagram was used. This gave an optimal range of $-0.1 < w_x < +0.1$, the same optimal range as was found for the infinite-weight version of the same problem. Figure 28 and Figure 30 show the mean sum squared error and smoothed normalized entropy plots for the data.

A summary of the results is given in Table 6. The benchmarking procedure again supplies the optimal lambda parameter where we have best odds of getting to the global error minimum. The point of optimal convergence for both epoch numbers turns out to be $\lambda = 6.3$, a similar result as for the same problem with infinite weights. At 1,000 epochs, the point of maximum entropy immediately precedes this optimal value, but at the much higher number of 25,000 epochs, the point of maximum entropy occurs at around $\lambda = 0.8$, a value significantly lower than the optimal learning rate parameter value. At 25,000 epochs a gap exists between $6.0 < \lambda < 6.3$ where all initial conditions converge to *S.S.E.* < 0.04.

Table 6. Results for training a finite weight 2:2:1 network on exclusive-or. Learning rate parameter range is $0 < \lambda < 50$ and initial weight condition range is $-0.1 < w_x < +0.1$.

EPOCH	MAXIMUM ENTROPY		MINIMUM MEAN *S.S.E.*		
	H	λ	MEAN	CONVERGENCE TO *S.S.E* < 0.04	λ
1,000	0.22	5.5	<4E-3	99/100	6.3
25,000	0.15	0.8	<6E-30	100/100	6.0 - 6.3

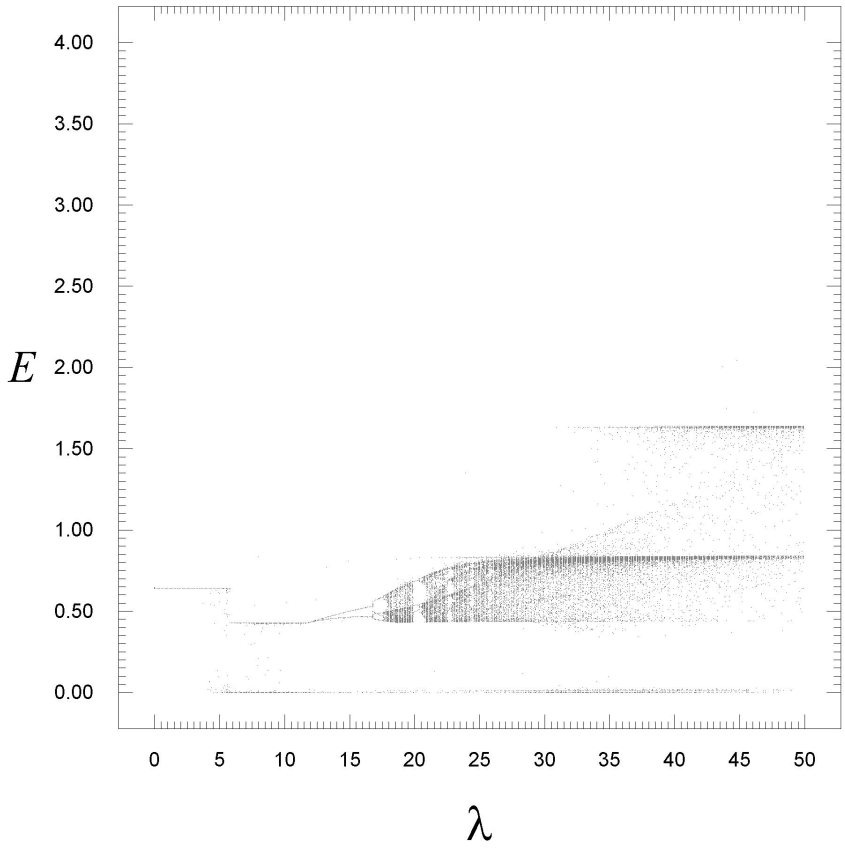


Figure 27. The sum squared error for the 2:2:1 biased, limited-feedforward network trained on the exclusive-or function and sampled at 1,000 epochs, plotted against λ. Target values of 0.9 and 0.1 are used to represent True and False. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 50$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 50,000,000 training epochs. No convergent gaps where $E <= 0.04$ were found.

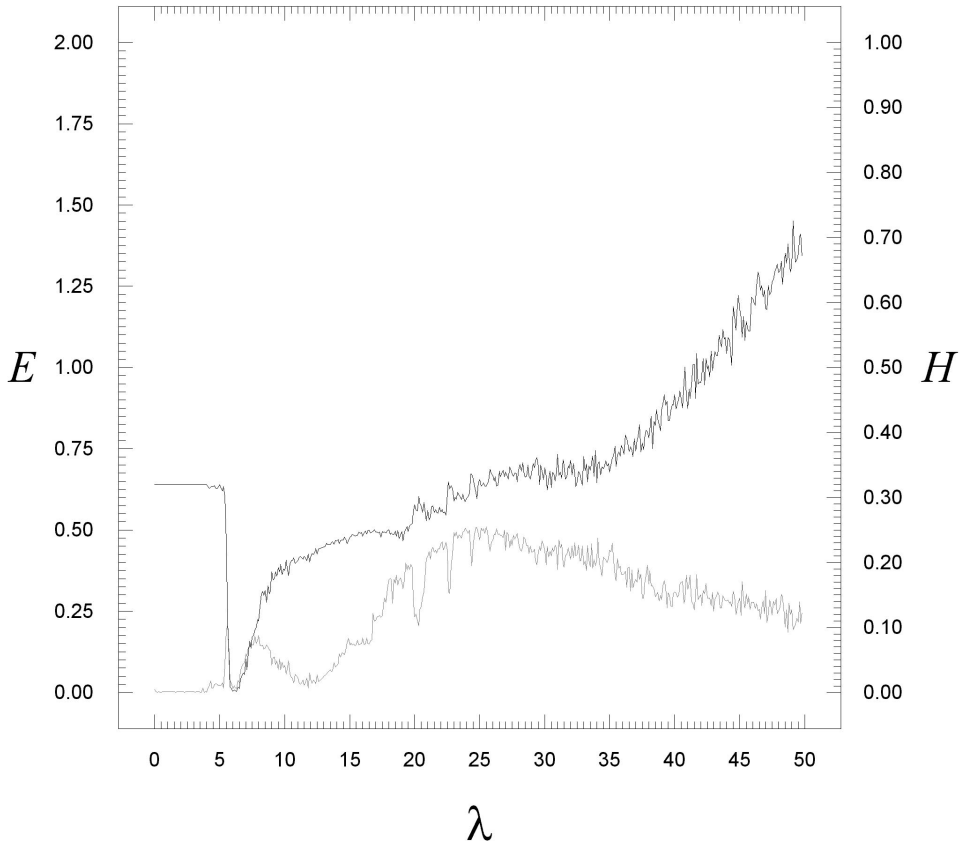


Figure 28. A comparison of the mean sum squared error *E* (black) with the entropy of the sum squared error *H* (grey), from the data shown in Figure 27. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of *E* is 1.45, at $\lambda = 49.0$.The first maximum of *H* is 0.22, at $\lambda = 5.5$. The minimum of *E* is 0.004, at $\lambda = 6.3$.

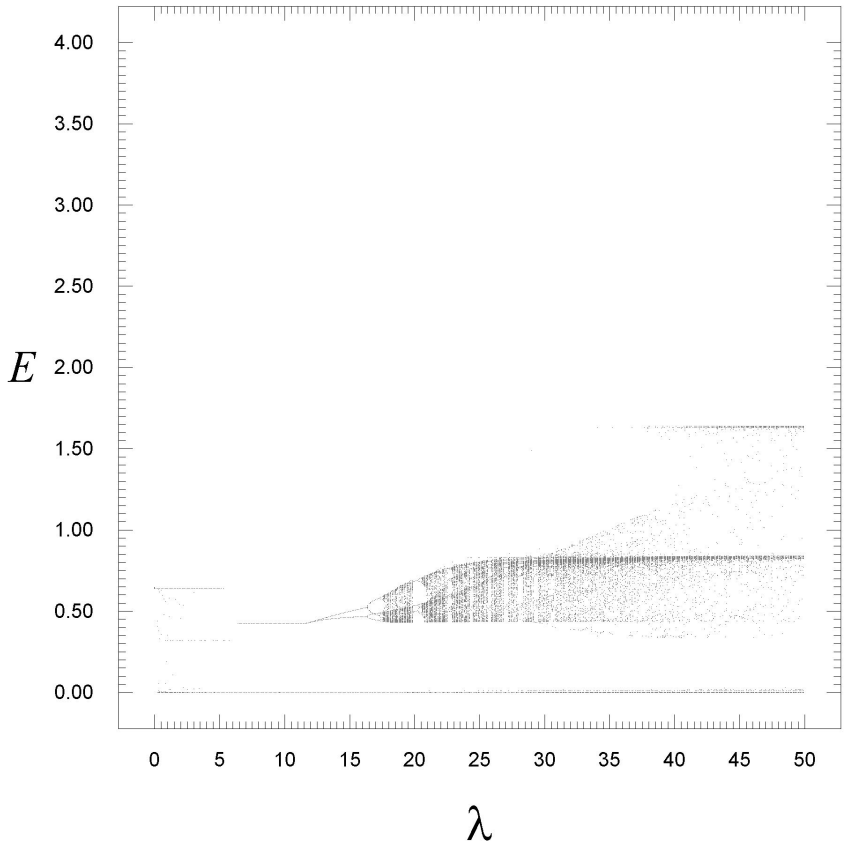


Figure 29. The sum squared error for the 2:2:1 biased, limited-feedforward network trained on the exclusive-or function and sampled at 25,000 epochs, plotted against λ. Target values of 0.9 and 0.1 are used to represent True and False. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 50$], using a step size of 0.1. Each step represents 100 different samples taken after initialising all weights in the range [$-0.1 < w_x < 0.1$]. This plot represents a total of 1,250,000,000 training epochs. A convergent gap was found where all *SSE* <= 0.04, between $6.0 < \lambda < 6.3$.

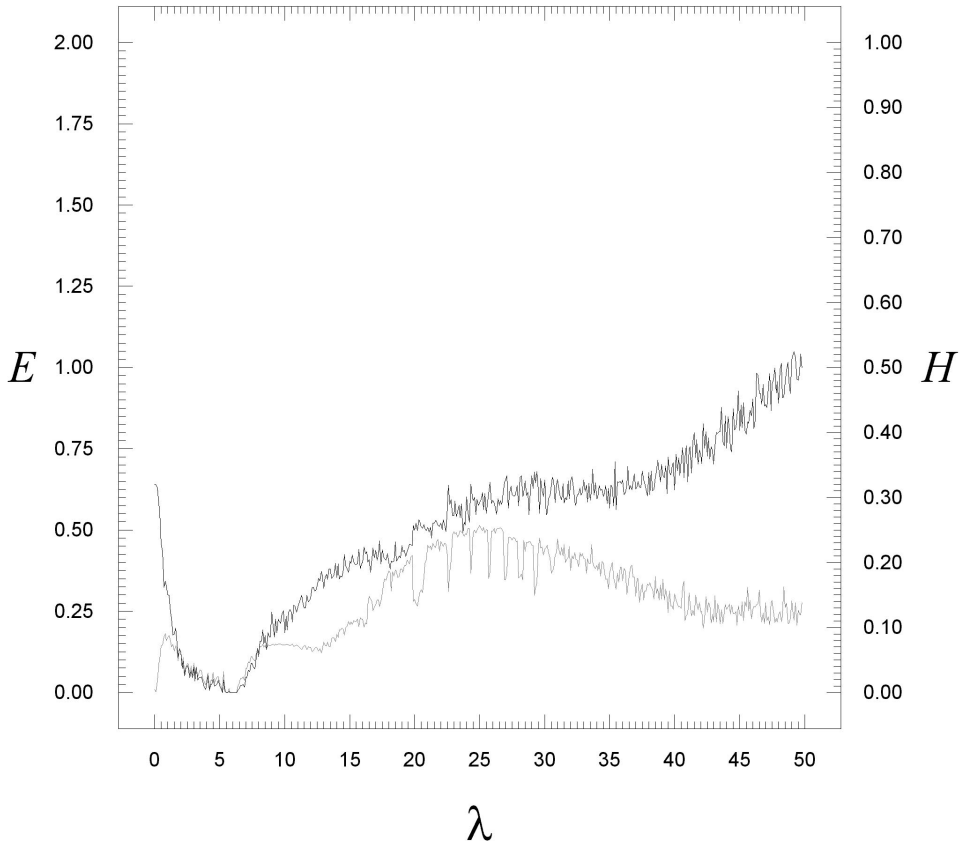


Figure 30. A comparison of the mean sum squared error *E* (black) with the entropy of the sum squared error *H* (grey), from the data shown in Figure 29. Entropy is calculated using 100 sample bins, and normalised to $\log_n(100) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range [$0 < E < 2$]. The maximum of *E* is 0.52, at $\lambda = 49.8$.The first maximum of *H* is 0.15, at $\lambda = 0.8$. The minimum of *E* is 6E-30, at $\lambda = 6.3$.

If we compare the mappings at 1,000 epochs for this finite-weight case (Figures 27 and 28) with the infinite-weight version of the same problem (Figures 25 and 26), we see that the mappings look very similar. At low values of the λ parameter the system has not converged. In this case we have target value of 0.1 and 0.9, so when all output values are 0.5 the resulting sum squared error is $4*0.4^2 = 0.64$. This is the *S.S.E.* value of the first solution line for low values of the λ parameter. This is a so-called temporary minimum, where the network has not been trained for enough epochs to separate the input patterns.

At the optimal value of $\lambda = 6.3$, the best convergence performance occurs, but there is still a possibility at 1,000 epochs that the system will reach a local as well as the global minimum. In the infinite weight case, this local minimum was at about *S.S.E.* = 0.66, in the finite-weight case it is at about *S.S.E.* = 0.42. So for both the infinite and finite weight case of the 2:2:1 topology learning exclusive-or at 1,000 epochs we can see that the problem is degenerate, in that there are no points along the optimal range of λ where one single solution forms. Instead, two solutions form, one a local minimum and the other the global minimum.

The situation where we have two possible solutions lasts until around $\lambda = 12$, at which point the solution line representing the local minimum begins to bifurcate. The period doubling cascade appears at a slightly lower parameter value than for the infinite weight case, where the bifurcation sequence began at about $\lambda = 14$. At the highest tested λ parameter we have three error levels available, where the sum squared error is either 0.0, 0.8, or 1.6. This again is very similar to the infinite weight case where at similar parameter values the corresponding minima were at sum squared error of 0.0, 1.0, or 2.0.

Figure 29 shows the bifurcation diagram for the problem tested at 25,000 epochs. This diagram shows a number of subtle differences to the diagrams produced with the much lower epoch numbers. A convergent gap was found with all *S.S.E.* < 0.04, between $6.0 < \lambda < 6.3$. Also, a new solution line appears with an *S.S.E.* of 0.32, between around $0 < \lambda < 6$. Lastly, we see that the global error minimum is always available no matter what the value of λ.

Figure 30 shows the mean sum squared error and entropy plots for the problem. Notable here is the location of the first maximum of *H*, occurring at $\lambda = 0.8$, preceding the optimal value of $\lambda = 6.3$ by a significant amount. This characteristic is different to the mappings of, for example, the 1:1:1 problem learning identity, where it was found that the first maximum of *H* clearly coincided with the start of the optimal value range for λ, independent of the epoch number. Also we see that the pronounced downward spike in both mean error and entropy of error at $\lambda \approx 6$ shown in all of the previous mappings for exclusive-or has widened considerably. These results suggest that for even higher epoch numbers, a relatively large gap of convergence to the global error minimum may well emerge, along a range of at least $0.8 < \lambda < 6.3$.

At the higher epoch number of 25,000 a new minima of *S.S.E.* = 0.32 emerges at very low λ, and disappears at the start of the globally convergent gap at around $\lambda \approx 6$. This minimum occurs when we have the correct output values for two input patterns, but the other two output values get stuck at 0.5, for example, the input/output transfer function becomes 00⇒0.1, 01⇒0.9, 10⇒0.5 and 11⇒0.5. This situation gives us a sum-squared error of $2*0.4^2 = 0.32$ when using targets of 0.1 and 0.9.

This minimum can occur when the weights from one input unit to both hidden units are both relatively high, so that when the input unit is on it turns on both hidden units. The weights from the hidden units to the output unit are arranged so that when both hidden units are on, the output unit gets a net input of zero. This leads to an output value of 0.5 for two of the input patterns. This local minimum is of the same type as that first described by Rumelhart, Hinton and Williams (1986a), who stated that they did not know the frequency of such minima, but that they were quite rare, and that the best way to avoid such minima is probably to use very small values of λ (Rumelhart, Hinton and Williams, 1986a).

Figure 31 shows the 2:2:1 topology. The local minimum type that we are considering here occurs when either w_{A1} and w_{A2} or w_{B1} and w_{B2} are both very high, causing both hidden units to become near saturation when an input is on. This type of local minimum, giving an identical error of *S.S.E.* = 0.32, has been reported by Lisboa and Perantonis (1991). The local minimum they describe has the weights from an input to both hidden units high enough for the input/output transfer function to become 00⇒0.5, 01⇒0.9, 10⇒0.5 and 11⇒0.9, resulting in the same sum squared error as the classic local minima reported by Rumelhart, Hinton and Williams (1986a).

This particular class of local minimum has been extensively analysed by Sprinkhuizen-Kuyper and Boers (1996b), who report that this type of minimum can be described as a saddle shape on the error surface, with a narrow central region that forms "a kind of rain gutter where the water can escape in some sink at the end" (Sprinkhuizen-Kuyper and Boers, 1996b, p. 15). They assert that this type of local minimum can be escaped if the weights from the inputs to the hidden unit that is near saturation can be made to tend to towards zero.

At the optimal value of $\lambda = 6.3$, the best convergence performance occurs, but there is still a possibility at 1,000 epochs that the system will reach a local minimum of *S.S.E.* = 0.42. At 25,000 epochs, the optimal value of $\lambda = 6.3$ ensures convergence to the global minimum, but as λ is increased past 6.3 the local minimum of *S.S.E.* = 0.42 becomes available again. The type of local minimum that becomes available at this point has again been studied by both Lisboa and Perantonis (1991) and Sprinkhuizen-Kuyper and Boers (1996b). The minimum comes about when the weights from both input units to one hidden unit are near saturation (in Figure 31, either w_{A1} and w_{B1} or w_{A2} and w_{B2} are both very high), causing only one of the four input patterns to be properly learned. All pattern presentations except for the one correct one will produce an output value of either 0.6333 or 0.3666.

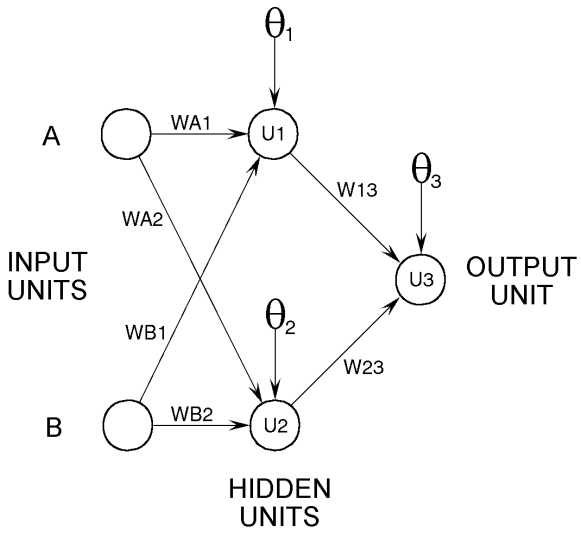


Figure 31. A 2:2:1 limited feedforward network topology, the same as shown in Figure 3. Labels are given for the hidden units, output unit and the nine independent weights.

We now have a description of the basic error minima types found for the classic 2:2:1 exclusive-or problem. The minima come about when the weights from the input units to the hidden units are high enough to cause one or more hidden units to be near saturation for one or more input pattern presentations.

From the analysis of the bifurcation diagrams produced in this and the preceding sections, we know that if we start a network off with low enough initial weight values, and train the network at an optimal λ parameter, we do not end up in any of these local minima. It would also seem likely that we can trade initial condition range for the epoch number, to a certain extent. For example, we may escape such minima given large initial weights, if we allowed a larger number of training epochs, and of course use the optimal λ parameter.

We can test this hypothesis using the high weight values reported in the literature for known local minima, training the network at the optimal λ parameter empirically determined in this section, and seeing if the network does escape the minima, given enough training epochs.

Table 7 shows the results from testing a number of known local minima for the 2:2:1 network learning exclusive-or. The first column of initial weight conditions is for the local minimum reported by Rumelhart, Hinton and Williams (1986a). The last three columns of initial conditions (labelled 1,2 and 3) are for similar local minima reported by Lisboa and Perantonis (1991). Case 1 is very similar to the minimum reported by Rumelhart, Hinton and Williams (1986a). The weights from an input unit to both hidden units are near saturation. Cases 2 and 3 are of the type where the weights from both input units to a hidden unit are near saturation.

The experiments used the standard generalised delta rule, output targets of 0.9 and 0.1 for true and false, and batch learning. Each experiment was set up so that the training was terminated when the sum-squared error was less than 0.04. One can see from Table 7 that the use of this optimal λ parameter does ensure convergence to the global error minimum in the long term, even given the high initial weight values. Additionally, the same experiments were performed with $\lambda = 1.0$. With the parameter set at this value, none of these minima were escaped, even after 20 million training epochs each.

Table 7. Results for training the 2:2:1 network of Figure 31 with target values of 0.1 and 0.9. The numbers in the bottom row show how many epochs are required to escape from each local error minimum at the optimal learning rate parameter of $\lambda = 6.3$.

		Rumelhart, Hinton & Williams	*Lisboa & Perantonis 1*	*Lisboa & Perantonis 2*	*Lisboa & Perantonis 3*
Initial Conditions	θ_1	2.0	12.59865	1.39427	-10.28005
	θ_2	-0.1	11.70733	5.121702	-10.97265
	θ_3	-0.8	1.54884	0.10897	-0.54656
	w_{A1}	-2.0	-11.52144	-13.70896	-10.42464
	w_{A2}	4.3	-11.89991	6.01491	-12.71414
	w_{B1}	9.2	-1.10568	-13.70896	8.55786
	w_{B2}	8.8	4.01044	6.01491	11.47785
	w_{13}	-4.5	4.59262	-3.42122	0.66331
	w_{23}	5.3	-6.14146	0.43758	4.23784
Initial SSE		0.320277	0.320016	0.426672	0.426682
Epochs To SSE < 0.04 @ $\lambda = 6.3$		89283	41358	82491	172504

8.5 Benchmarks for the 2:1:1 net Learning All Binary Boolean Functions

The 2:1:1 biased and fully connected feedforward network has, in principle, all the topological and computational resources to learn any possible binary Boolean function. The network was therefore tested across all the 16 possible Boolean functions from Table 2. This gives us a test of convergence dynamics that provides a complete systematic coverage for these functions. The resulting errors are not added together but each is recorded as a separate point.

Three simulation runs were performed, at 100, 250 and 1,000 epochs, in the range $0 < \lambda < 100$. Again, initial range for weight values was calculated by the method of using progressively smaller initial conditions until we arrive at a bifurcation map showing the minimum number of solutions. This gave an initial condition range of $-0.01 < w_x < +0.01$. The bifurcation maps for the data collected from the tests are shown in Figures 32, 34 and 36, and the associated mean error and entropy of error plots are shown in Figures 33, 35 and 37, respectively.

Considering the first map at 100 epochs (Figure 32) we can clearly see bifurcation effects starting at around $\lambda = 6$ and continuing until about $\lambda = 30$. The plot of the mean error (Figure 33) shows that there is a minimum of mean error at $\lambda = 11.5$, which is inside this region. The bifurcation effects are from some of the functions to be learned, and not others.

At 100 epochs there were no discrete gaps showing convergence to the global error minimum. However, this can be found with an increased number of epochs. The map for the same experiment at 250 epochs is shown in Figure 34, with the associated mean and entropy map in Figure 35. There is a clear convergent gap where all trajectories converge to a *S.S.E.* of < 0.04, occurring in the range $5.4 < \lambda < 18.2$.

Because we are grouping 16 experiments together, each comprised of 100 data points, there are a large number of bins for the entropy measurement. The 1600 bins therefore represent very small subintervals of the sum squared error, and so our measure of *H* will only reach zero when all the points converge to within an exceedingly small range, which is very unlikely. This is why we do not see a corresponding gap of zero entropy for the gap where all *S.S.E.* measurements are less than 0.04.

Figure 36 shows the bifurcation map and Figure 37 shows the mean error and entropy for the same experiment tested at 1,000 epochs. The increased iterations do not affect the end position of the convergent gap at $\lambda = 18.2$ but bring forward the start of the gap to $\lambda = 4.6$, compared to $\lambda = 5.4$ at 250 epochs. The bringing forward of the start position of the gap is probably caused by allowing enough epochs for temporary minima to reach convergence.

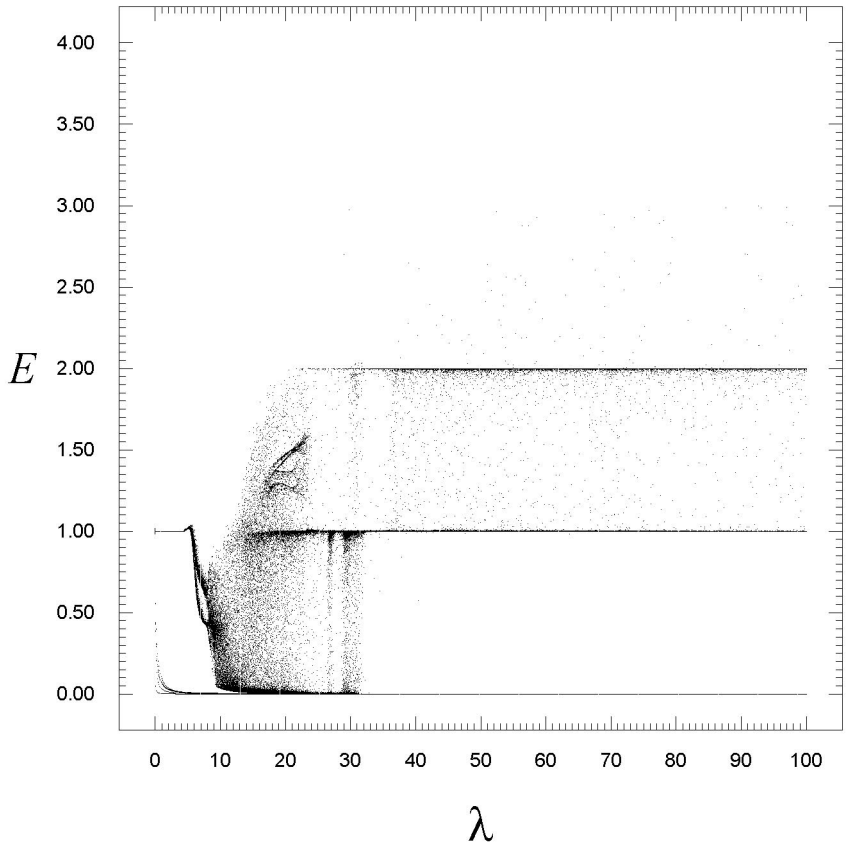


Figure 32. The sum squared error for the 2:1:1 biased feedforward network trained on all the binary Boolean functions sampled at 100 epochs, plotted against λ. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 100$], using a step size of 0.1. Each step represents 1600 different samples taken after initialising all weights in the range [$-0.01 < w_x < 0.01$]. This plot represents a total of 160,000,000 training epochs. No convergent gaps where all *S.S.E.* <= 0.04 were found.

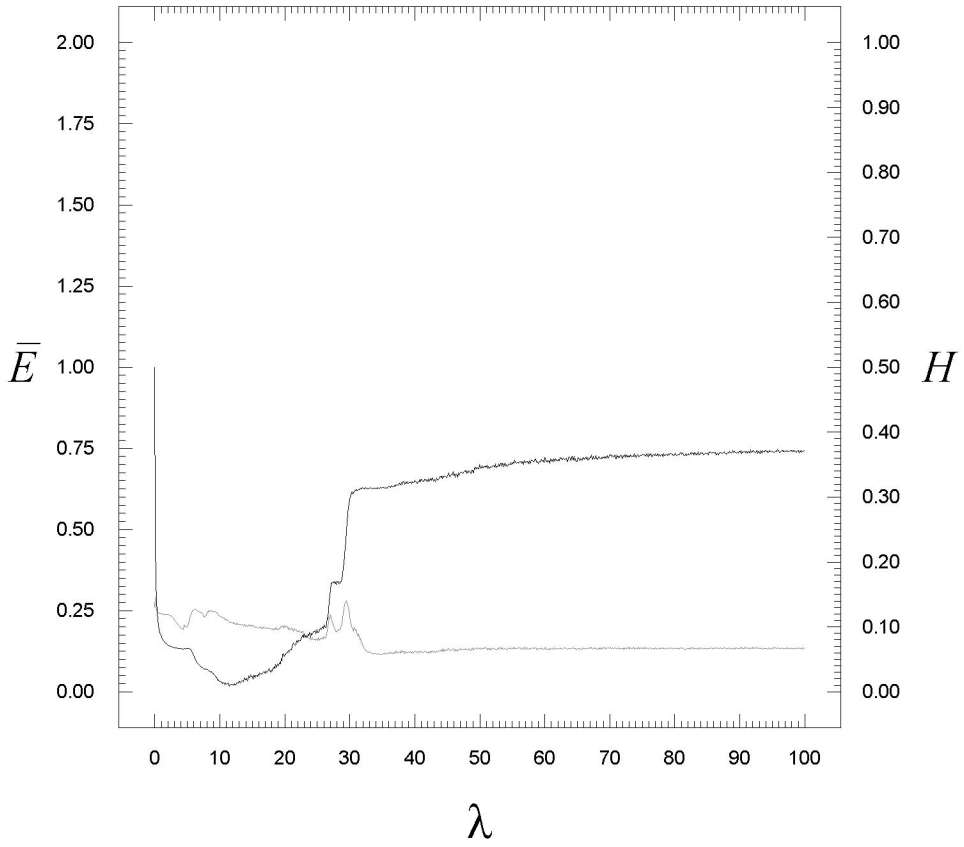


Figure 33. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 32. Entropy is calculated using 1600 sample bins, and normalised to $\log_n (1600) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of E is 1.000, at $\lambda = 0.0$. The maximum of H is 0.289, at $\lambda = 0.1$. The minimum of E is 0.017, at $\lambda = 11.5$.

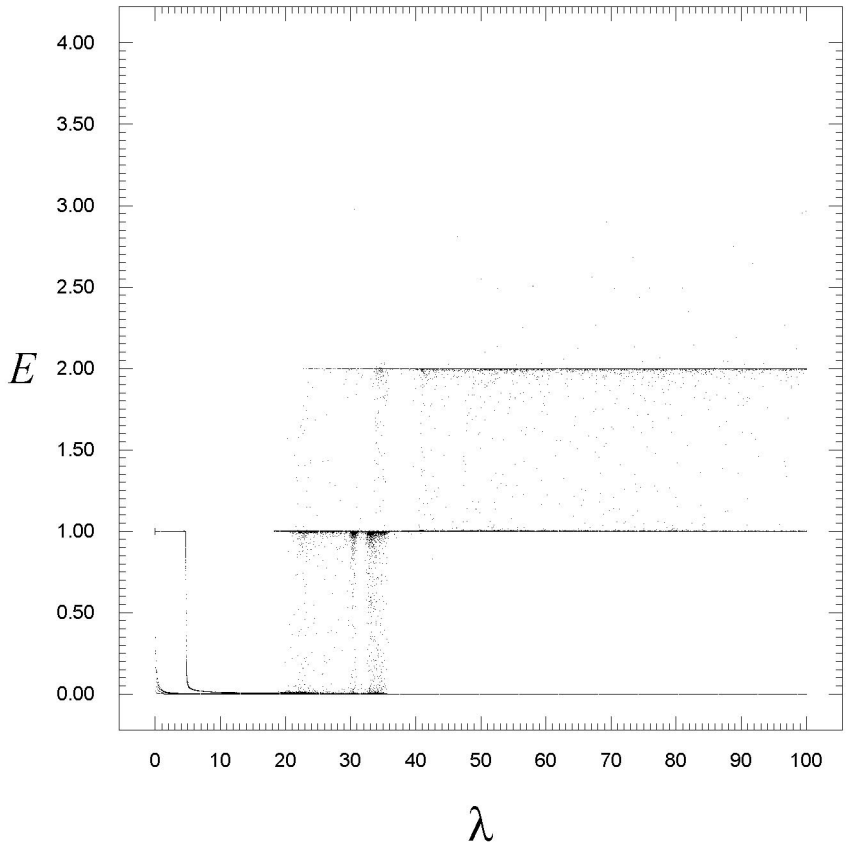


Figure 34. The sum squared error for the 2:1:1 biased feedforward network trained on all the binary Boolean functions sampled at 250 epochs, plotted against λ. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 100$], using a step size of 0.1. Each step represents 1600 different samples taken after initialising all weights in the range [$-0.01 < w_x < 0.01$]. This plot represents a total of 400,000,000 training epochs. The convergent gap where all *S.S.E.* <= 0.04 starts at $\lambda = 5.4$ and ends at $\lambda = 18.2$.

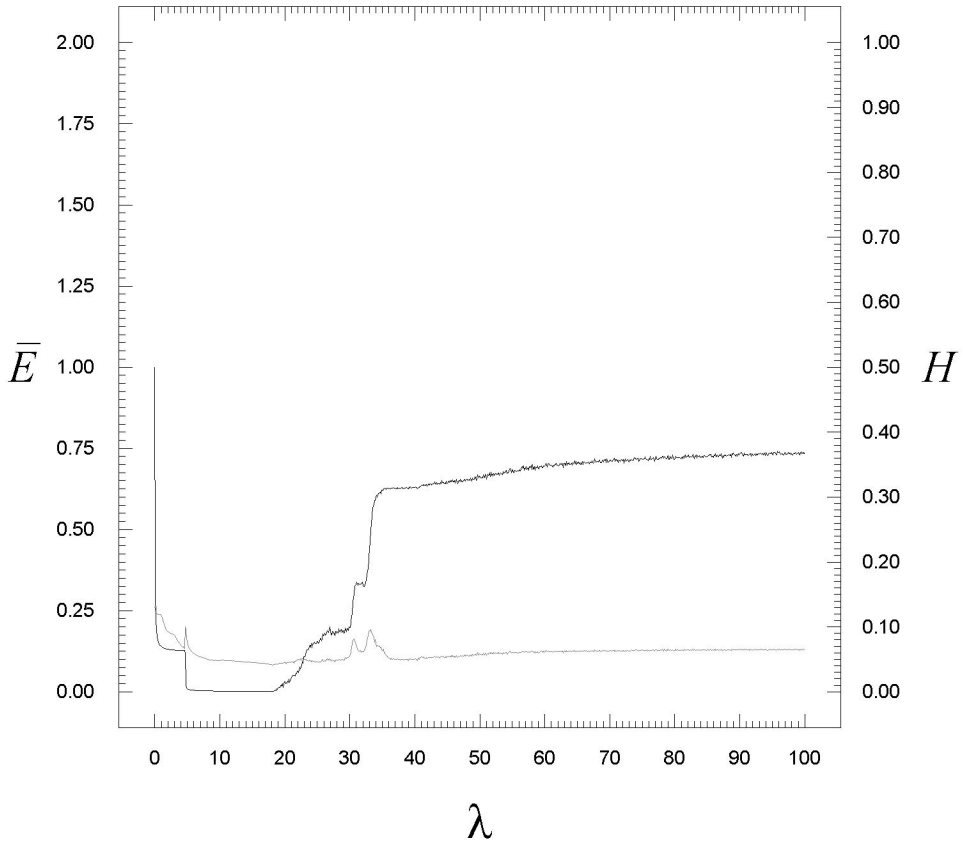


Figure 35. A comparison of the mean sum squared error E (black) with the entropy of the sum squared error H (grey), from the data shown in Figure 34. Entropy is calculated using 1600 sample bins, and normalised to $\log_n(1600) = 1$. The smoothing factor is 100. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of E is 1.000, at $\lambda = 0.0$. The maximum of H is 0.261, at $\lambda = 0.0$. The minimum of E is 0.001, at $\lambda = 18.2$. The minimum of H is 0.082, at $\lambda = 18.2$.

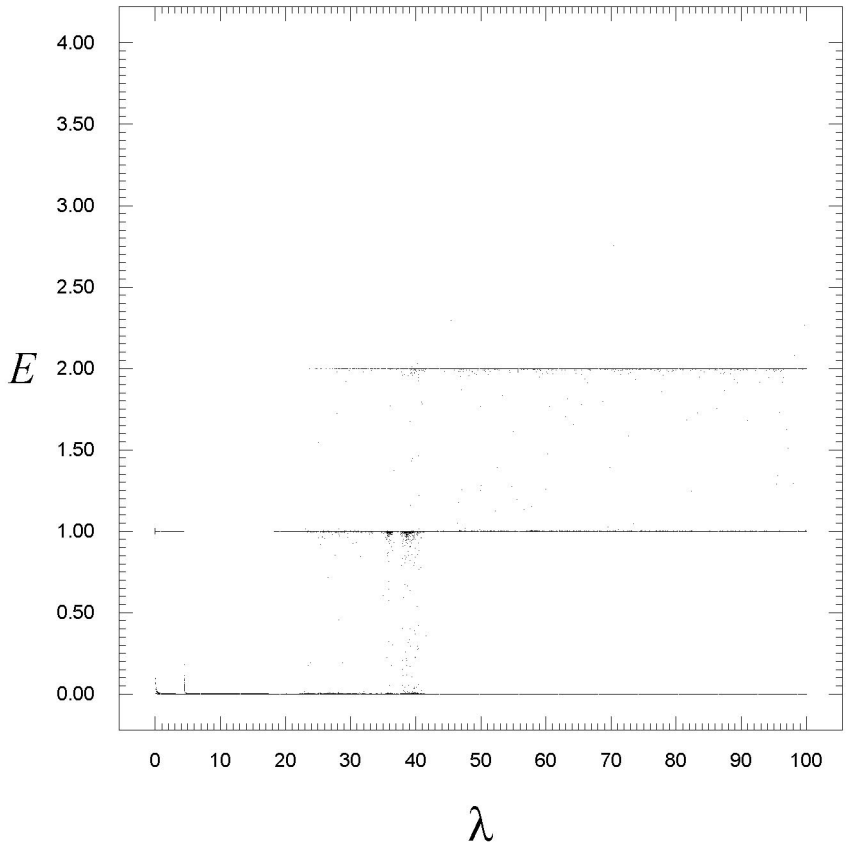


Figure 36. The sum squared error for the 2:1:1 biased feedforward network trained on all the binary Boolean functions sampled at 1,000 epochs, plotted against λ. Sum squared error is in the range [$0 < E < 4$] and λ is varied over the range [$0 < \lambda < 100$], using a step size of 0.1. Each step represents 1600 different samples taken after initialising all weights in the range [$-0.01 < w_x < 0.01$]. This plot represents a total of 1,600,000,000 training epochs. The convergent gap where all *S.S.E.* <= 0.04 starts at $\lambda = 4.6$ and ends at $\lambda = 18.2$.

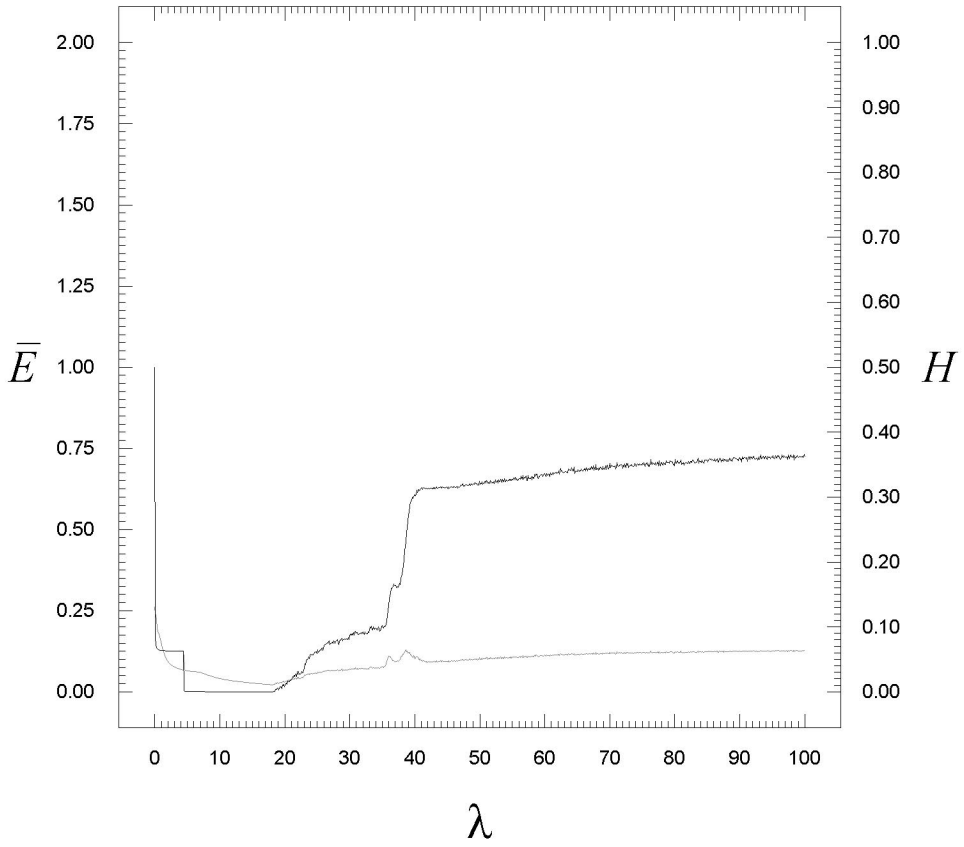


Figure 37. A comparison of the mean sum squared error *E* (black) with the entropy of the sum squared error *H* (grey), from the data shown in Figure 36. Entropy is calculated using 1600 sample bins, and normalised to $\log_n (1600) = 1$. Mean sum squared error is plotted in the range $[0 < E < 2]$. The maximum of *E* is 1.000, at $\lambda = 0.0$. The maximum of *H* is 0.262, at $\lambda = 0.0$. The minimum of *E* is 0.000, at $\lambda = 18.2$. The minimum of *H* is 0.022, at $\lambda = 18.2$.

Performing the statistical analysis at 1,000 epochs shows that the points of minimum of entropy of error and of minimum mean *S.S.E.* across all the binary Boolean functions are coincident at $\lambda = 18.2$, where $H = 0.022$ and mean *S.S.E.* = 0.0.

The data for the 1,000 epoch experiment were then split up for individual analysis of each Boolean function. The results of this breakdown are given in Table 8. One can see that the overall convergent gap of $4.6 < \lambda < 18.2$ is a composite of the overlapping convergence gaps of all the individual functions.

Table 8. Complete sum-squared error convergence statistics for a 2:1:1 biased feedforward network learning all the binary Boolean functions, taken at 1,000 epochs. Initial weight values are in the range $-0.01 < w_x < +0.01$. Refer to Table 2 for a truth table for these functions.

	CONVERGENT GAP (*SSE*<0.04)	
f#	λ_{START}	λ_{END}
0	0.100	100.0
1	0.200	35.20
2	0.200	37.40
3	0.100	100.0
4	0.200	37.50
5	0.100	100.0
6	4.600	18.20
7	0.200	24.60
8	0.200	24.50
9	4.600	18.20
10	0.100	100.0
11	0.200	37.60
12	0.100	100.0
13	0.200	37.40
14	0.200	35.40
15	0.100	100.0
ALL	4.600	18.20

The complete set of bifurcation and mean error/entropy of error mappings for each individual function shown in Table 8 are given in Appendix A. These mappings can be grouped in order of stability, with respect to the lambda parameter – the function having the widest convergence gap being the most stable.

All of theses mappings are shown overlaid together in Figure 38. It was found that the regions of global convergence overlap from $4.60 < \lambda < 18.20$ at 1,000 epochs. The gap is discrete, empirically demonstrating that global minimum error convergence to any binary Boolean function across this range is achieved with a probability of 1. This, I believe, is a remarkable finding, having found no such demonstration elsewhere. There is no evidence that this should hold for non-Boolean functions.

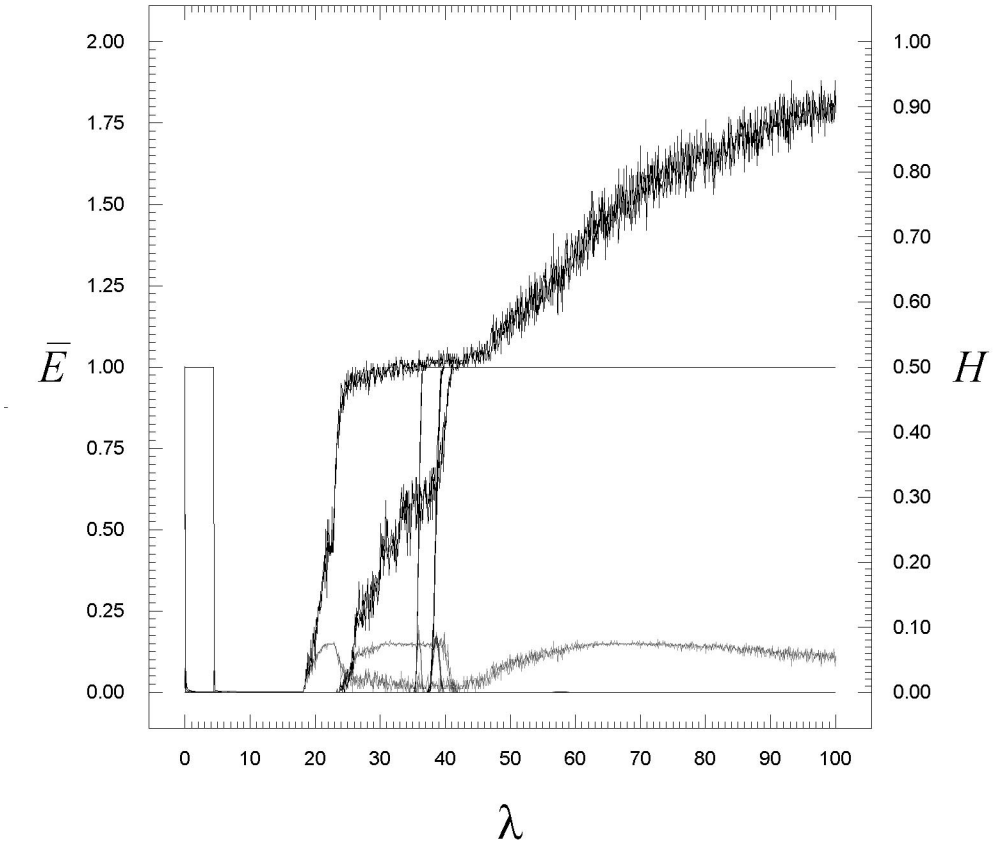


Figure 38. A composite of all individual mappings of mean sum squared error E (black) with the entropy of the sum squared error H (grey), as a function of λ for a 2:1:1 network trained across all the binary Boolean functions at 1,000 epochs. The convergent gap across the entire data set where $S.S.E < 0.04$ is between $4.6 < \lambda < 18.2$. The sum squared error statistics for each individual function are given in Table 8.

Listing the 16 mappings in order of stability with respect to the lambda parameter gives five characteristic individual types of dynamical performance showing different globally convergent gaps where *S.S.E.* < 0.04.

The most stable group are TRUE, FALSE, A, ~A, B, ~B. The corresponding mappings for these problem-types are all characterised by completely flat mappings (i.e., the error is constant zero no matter what the value of λ), therefore being the most stable possible with respect to λ.

The next most stable group were A AND ~B, ~A AND B, A OR ~B, ~A OR B. The corresponding mappings for these problem-types are all flat up until about $\lambda = 37.5$, where after this range only a local minimum of *S.S.E.* = 1.0 is available.

The next two functions, AND and NAND, were slightly less stable, with the convergent gap ending at around $\lambda = 35.3$. Even less stable was the next group, the functions OR and NOR. These two functions have a global convergence gap up to about $\lambda = 24.6$ and 22.2, respectively. The mappings show that there is a transition after this point to two solutions, the global minimum, and a local minimum. The frequency of convergence to the local minimum increases until around $\lambda = 40$, at which the local minimum becomes the only available solution.

The last and most difficult problems are XOR and XNOR. The mappings for these functions are distinguished from all the others in that the gap of global convergence does not occur at low lambda values. Instead, the network converges to a local minimum of *S.S.E.* = 1.0 until the parameter is increased to 4.6, and the global convergence gap lasts until $\lambda = 18.2$. After this, both the global and the local error convergence modes become available again. The fact that the gap of convergence does not begin until $\lambda = 4.6$ could well be due to the network not having had enough epochs to separate the input patterns, so we are seeing a temporary minimum here.

After λ is increased past about 30-40, the global minimum is unavailable and instead only two local minima of *S.S.E* of 1.0 and 2.0 are available. The mean error slowly climbs to 2.0 as the entropy falls off, showing that the local minimum *S.S.E*. of 2.0 becomes the highest frequency mode.

It was unexpected to find that the AND and OR operators were most stable when one input was complemented. It would seem that generalised delta rule, at least in this case, prefers asymmetrically complemented inputs to symmetrical ones. This could perhaps be due to different error terms. Another unexpected result was that AND and NAND were found to be more stable with respect to the lambda parameter (and thus easier to learn) than OR and NOR.

Chapter 9

Conclusions

This thesis examines the dynamical properties of the generalised delta rule for Boolean problem solving using a number of small networks. A method is developed throughout the thesis to determine ranges of initial conditions and learning rate parameters for optimal learning performance.

The pilot study proved the effectiveness of using a modified form of the standard bifurcation diagram to evaluate learning using the generalised delta rule. The bifurcation diagram presented here is a scatter plot of the error measured at the last epoch of a network being tested. Error measurements are taken for a number of different values of the learning rate parameter, and each different learning rate parameter is tested a number of times across a range of initial conditions. Because we are measuring the performance error, the number of weights used in the network becomes redundant. This type of mapping does partly address the need for a visualisation of the error surface irrespective of dimensionality, a requirement also noted by Chang and Mak (1999).

The results presented in the pilot study showed that the bifurcation diagram method can distinguish the minima of an error surface for a problem utilising the generalised delta rule. The diagrams can also discern input pattern presentation order effects, and period doubling effects such as those described under universality theory proposed by Feigenbaum (1980). With universality theory comes the promise of limited predictability for the long-term behaviour of the generalised delta rule, and since under universality theory, dimensionality is also irrelevant, the same geometrical property quantitatively determined by the number F will work for systems with any number of weights.

The pilot study first concentrated on the saddle-shaped error surface of the 1:1:1 network learning the Boolean identity function. The local minimum on this error surface was found to be a true minimum, in that points initially within the domain of this minimum could not escape, even using on-line training. This result is contrary to the suggestion by Sprinkhuizen-Kuyper and Boers (1996a,b) that saddle-shaped regions of the error surface are not true minima as they should be escapable using the on-line training method.

A standard benchmark was then developed, whereby a plot of the mean of the sum squared error values at a specified epoch are shown alongside a plot of the entropy of those values, both as a function of the learning rate parameter λ. This type of diagram was found to have some very useful characteristics, in particular the ability to determine convergence for any number of solutions to a given problem.

The standard benchmarking method was used to determine a range of initial conditions and learning rate parameter for the 1:1:1 network trained on the Boolean identity problem using batch learning. A region was found at very high parameter values where points initially in the domain of the local minimum were escaped, the parameter range thus ensuring convergence to the global minimum of the error function.

The globally convergent range for this 1:1:1 problem showed a character unlike the other problems benchmarked in this study. Notably, the start of the range with reference to the λ parameter was found to be coincident with the point of maximum entropy. The graphical analysis of this problem showed that the learning trajectories within the optimal range are maximally disordered, and partly ergodic, becoming independent of the initial conditions. Within the optimal range for the λ parameter the basin structure breaks up, with the previously separate attractors corresponding to the two different sets of initial conditions joining together to form a single global attractor.

The analysis showed that when using low values of the λ parameter, the system can only converge to the nearest error minimum from the initial conditions. At a high range for the parameter, the system can become partly independent of the initial conditions, and so escape the local minimum. This result would seem to be an example of what Kolen and Pollack (1990) were suggesting when stressing that "methods which harness 'strange attractors' ergodically guaranteed to come arbitrarily close to some subset of solutions might work better than methods based on strict gradient descent." (p. 272). This study found this result with the standard 1:1:1 topology learning the Boolean identity function with the generalised delta rule. This example also shows that using very low values of the λ parameter, suggested by Rumelhart, Hinton and Williams (1986a) for escaping local error minima, is not necessarily true.

The results for benchmarking the 2:2:1 topology with the exclusive-or problem with both infinite and finite weight solutions showed a number of similar dynamical characteristics. A number of local minima were found, firstly the existence of temporary minima at low values of the λ parameter. The existence of this effect, characterized by an equality of outputs corresponding to all training patterns, is in agreement with a number of the conclusions of other authors, notably Liang (1991) and Lisboa and Perantonis (1991). Ampazis, Perantonis and Taylor (1999) have proposed a method for minimizing the time spent in such minima.

The 2:2:1 exclusive-or problem was found to have an optimal learning rate parameter value of $\lambda = 6.3$, a similar value found for both the finite and the infinite weight version of the problem. The benchmarking procedure showed a narrow downward spike in the mean sum squared error and entropy of error was found at the optimal parameter value. By benchmarking the problem at much higher epoch numbers, it was found that the optimal parameter range widened with increasing training epochs.

The point of maximum entropy was not found to coincide with the start of the globally convergent gap for the 2:2:1 exclusive-or when tested at a high epoch number, suggesting a different dynamical convergence mechanism to that found for the 1:1:1 network trained on the identity problem. Because the point of maximum entropy was quite low ($\lambda < 1$) at the largest epoch number tested, we may tentatively suggest that in the long term, there may be a region of global convergence for this problem between at least $1 < \lambda < 6.3$.

The 2:2:1 exclusive-or problem also highlighted the condition of degeneracy, whereby a particular range of the λ parameter allows two possible long-term solutions, one a local minimum and the other the global minimum. At the optimal value for the λ parameter, there is a higher likelihood of global convergence with higher epoch numbers, suggesting that the generalised delta rule can therefore bring the system out of degeneracy, in the long-term.

Two specific local minima types for this problem were identified and analysed, both occurring when the weights from the input units to the hidden units are relatively high, forcing one or both hidden units near saturation for certain input patterns. Examples of these specific minima can be found in the literature. The first type belongs to a class of minima first reported by Rumelhart, Hinton and Williams (1986a), who stated that they did not know the frequency of such minima, but that they were quite rare, and that the best way to avoid such minima was probably to use very small values of λ. The methods presented in this thesis provided a means of mapping the frequency of such minima, for example showing that the classic 2:2:1 minimum first reported by Rumelhart, Hinton and Williams (1986a) is of relatively low frequency, occurring after a relatively high number of epochs, where $0.0 < \lambda < 6.0$. The second type of local error minimum found in this study has also been reported by Lisboa and Perantonis (1991). The results determined empirically that in fact the best way of avoiding both these specific types of minima is to use much higher values of the λ parameter than the original authors suggested, a value of $\lambda = 6.3$ being optimal for the 2:2:1 exclusive-or.

The results from testing the 2:2:1 exclusive-or problem showed that an optimal value for the λ parameter and a correct range of initial conditions will ensure that these minima are avoided altogether within a finite number of iterations. By testing the optimal parameter on known error minima reported by Rumelhart, Hinton and Williams (1986a) and Lisboa and Perantonis (1991), it was shown that the system will eventually converge to the global minimum of the error function, even given initial weights that are from such a high range as to drive some units to near saturation. From this, we may tentatively suggest that this optimal λ parameter must allow the weights that are forcing a unit to near saturation to tend to zero faster than otherwise. Also, the fact that the parameter worked with such large weight values suggests that we may be able to "trade" large initial condition ranges for increased iterations, ending up with the same result.

This study provides a way of benchmarking the dynamics of learning for the generalised delta rule. The example of benchmarking all the binary Boolean functions for the 2:1:1 biased feedforward network shows how this can be done in a complete and systematic manner. The fact that a discrete band-gap of global minimum error convergence was found to exist across *all* the binary Boolean functions was a truly remarkable result. This kind of dynamical behaviour is, as far as I am aware, hitherto unknown. I can find no other references in the literature where a similar experiment has allowed the observation of this behaviour. The results show that at 1,000 epochs the convergent gap across the entire set of binary Boolean functions is between $4.6 < \lambda < 18.2$, for *all* initial conditions in the range $-0.01 < w_x < +0.01$.

The results did throw up some interesting agreements and disagreements with the work of Sprinkhuizen-Kuyper and Boers (1996a,b) who have asserted that, for the exclusive-or problem trained using both the 2:1:1 and the 2:2:1 topologies, there are no local minima. The reasoning for this claim is that all the minima for these problems are saddle-shaped regions of the error surface. From this result it can be concluded that from each finite point in weight space a strictly decreasing path exists to a point with error zero (Sprinkhuizen-Kuyper and Boers, 1996b).

When a saddle point is encountered, the on-line training method can, they claim, probably escape from such a saddle point, since the error surface is not horizontal for each individual pattern, only the average error surface for all patterns (i.e., the error surface "seen" by batch learning) is horizontal. Using on-line training, "a small change of the weights in the right direction will decrease the error, moving away from the saddle point" (Sprinkhuizen-Kuyper and Boers, 1996a).

The results from the pilot study of this thesis show that on-line training does *not* escape from the simplest possible saddle-shaped error surface, the 1:1:1 network learning the Boolean identity function, certainly at low values of the parameter λ. This study shows saddle-shaped regions are therefore true local minima, with respect to batch or on-line training, but not necessarily with respect to certain values of the parameter λ.

The entropy and mean error mappings do seem the most useful of benchmarks for the study of learning convergence. With these mappings we can determine solution modes, whether discrete band-gap, as is the case for the 1:1:1, or the 'dip' in relative solution frequencies such as is the case for the 2:2:1 exclusive-or, the most degenerate problem type. The optimal value for the parameter λ in this case is impossible to spot from the bifurcation diagram alone.

Gradient descent is a local method. It is apparent that the networks at low lambda parameter values will converge well, but will always converge to the next lowest local minimum on the error surface. The optimal parameter ranges found empirically by the methods developed in this thesis always converged to the global minimum. The fact that guaranteed global error convergence was found at surprising high lambda parameter values suggests that this is the very reason why the factor was originally overlooked in some earlier research. Most researchers were trying to avoid unstable behaviour by keeping lambda values low. This study shows that this viewpoint can be mistaken – regions of optimal global convergence can be found with high parameter values. Ritter (1991) showed that high values of lambda in machine learning can help find solutions, evidence that supports this result.

9.1 Further Work

The benchmarking method developed throughout this thesis would have many possible further applications. Benchmarking the problems presented here at even greater parameter values, and benchmarking much larger networks are a possibility. The network sizes used in this thesis are exceedingly small compared to contemporary networks, so benchmarking much larger networks could show if scaling is likely to be a problem. The method could also be applied to non-Boolean problems and recurrent networks.

A momentum term is sometimes included in simulations that use the generalised delta rule, in order to speed up the algorithm. Qian (1999) has demonstrated an analogy between the momentum term in gradient descent and the mass of Newtonian particles in a conservative force field. Using this analogy, it was found that the momentum term can not only increase the speed of convergence, but can also nearly double the parameter range for convergence. The benchmarking method could be used to empirically verify this hypothesis.

A method known as 'line-search' exists (Kramer and Vincentelli, 1989) to determine an optimal learning rate parameter whilst training is in progress. Determining such an optimal lambda parameter can be regarded as a one-dimensional optimisation problem, where in the simplest case, a small initial learning rate is used, which is iteratively increased until the error-function no longer decreases (Riedmiller and Braun, 1993). Such adaptive techniques can also be benchmarked using the method set forth in the thesis, by plotting sum-squared error etc., as a function of the initial value for the parameter λ.

9.2 Speculation

Throughout this study I have concentrated on artificial neuro-dynamical models that highlight the theoretical and empirical problems that transpire when we use the standard generalised delta rule, as proposed by Rumelhart, Hinton and Williams (1986a,b). We know that the generalised delta rule was developed specifically as a non-linear learning procedure. This study has empirically shown that the procedure is tractable under the universality theory of Feigenbaum (1980), precisely because of this nonlinearly. Universality makes possible precise predictions concerning the limiting behaviour of the procedure, for example, the ability to predict the bifurcation points using the number *F*, and has allowed us to harness and apply the statistical properties of entropy and ergodicity to the gain an overview of the dynamics of the procedure.

The concentration on the problem of the 1:1:1 biasless network learning the identity function brought into focus the problem of local error minima, and the solution to it. The results show that there is a band-gap in the dynamics of this problem, where all learning trajectories universally converge to the global minimum of the error function, when the parameter λ is varied between 100 and 180. The actual width of the gap with respect to the lambda parameter is attributable to universal scaling, since the gap's start transition occurs when the lambda parameter is sufficiently high enough to reach the maximal entropy required for the system to escape the initial conditions.

Universality theory can describe the possible routes the system can take. Entropy is related to the loss of information as to initial conditions, and at maximum entropy, the system is independent of the initial conditions, and the behaviour is said to be ergodic. In one sense, the initial conditions versus the lambda parameter are interchangeable or equivalent to each other. This can happen where a certain lambda parameter "tunes" the system into a highly ergodic mode. The higher the ergodicity, the more the system behaviour becomes independent of the initial conditions.

The effect of tuning the lambda parameter for optimal odds of getting to the global error minimum was shown clearly in the example of the simple 2:2:1 limited-feedforward network learning the exclusive-or function. The results show that a λ parameter value of 6.0 will give the best odds of converging to the global minimum of error. The higher likelihood of the frequencies of global convergence at higher epoch numbers with this degenerate problem type also suggests that the properties of the generalised delta rule bring a system out of degeneracy, in the limit.

The mechanism of tuning a neural network so that it is guaranteed to learn solve a certain problem could be a very useful concept to apply to psychology. For example, why is it that everybody is guaranteed to learn, amongst many things, their mother tongue? In linguistics, this phenomenon is sometimes called universal language acquisition (not to be confused with the universality theory of Feigenbaum, 1980) and is partly rooted in the theory of linguistic structure put forth by Chomsky (1957). Experimental work such as that described in this thesis could provide insight as to how real brains might be tuned (by evolution?) to be guaranteed to learn certain things. Similarly, the critical period for language learning might be explained by a decreasing lambda.

Taken together, these results demonstrate a kind of empirical convergence proof for the generalised delta rule procedure using relatively low initial condition ranges and relatively high lambda parameter ranges, for at least the problems considered. I suggest that the procedure is capable of learning any combinatorial logic function in this manner. It is good that it turns out that we can guarantee global minima convergence with the completely standard generalised delta rule procedure, or ‘vanilla flavoured back-propagation’, as it is sometimes called, without any ‘improvements’. Specifically, the route to convergence has been shown to be tractable under universality theory and so should be independent of both the actual activation function (so long as it is non-linear) and the initial weight conditions (to within a specified range). Whether these results will hold for larger networks is an open question.

The simulations in this study were written in a number of languages and run on different machines, in an attempt to see if the results were dependant on the floating point representation of a specific machine or language. The results of running these simulations in Basic, C, Lisp, and machine code on a number of processors all generated essentially identical results. Non-linear iterative learning systems should be tractable under universality theory even if the underlying numerical representation gives the system a large round-off error, such as fractional encoding using sixteen-bit arithmetic, or even less accurate schemes.

The methods set forth in this study provide an empirical way of locating optimal properties by producing zoom-in views of the total dynamical structure found in any generalised delta rule system. These empirical methods are, in effect, experimental protocols for discovering optimal parameters and conditions with respect to the convergence to the global error minimum of any generalised delta-rule system. The benchmarking procedures as they stand allow us to explore the dynamics of any generalised delta-rule system, and the procedures are complete.

Rumelhart, Hinton and Williams (1986a,b) admitted that their procedure could not guarantee a solution for all solvable problems. Minsky and Papert (1988) have, in their criticism of the approach of Rumelhart, Hinton and Williams (1986a,b), suggested that the non-linear squashing function added nothing to the original perceptron convergence procedure. This study shows that Minsky and Papert may have been misplaced in their criticism of the non-linear generalisation. In fact, the opposite seems to be more the reality – the linear perceptron convergence procedure has been replaced by a much more elaborate and powerful procedure that can also be guaranteed to find an optimum set of weights, if that set of weights exists.

This empirical study does clearly show that the non-linear delta rule as described by Rumelhart, Hinton and Williams (1986a,b) is guaranteed to find the solution, if the set of weights exist, at least for the small range of problems described in this thesis. Surprisingly, this was found at high lambda-parameters, and has been shown as an effect of universality – precisely because of the squashing function.

These findings negate the suggestion by Minsky and Papert (1988) that the squashing function adds nothing to the old linear perceptron rule. Although this convergence demonstration is currently only empirical, I would hazard a guess and say that it is only a matter of time before a fully analytic treatment of universality theory is uncovered, and at that point a truly analytic convergence proof for the generalised delta rule will be possible.

References

Ampazis, N., Perantonis, S. J., and Taylor, J. G. (1999) *Dynamics of Multilayer Networks in the Vicinity of Temporary Minima.* Neural Networks **12**, pp. 43-58

Baker, G.L., and Gollub, J.P. (1990) *Chaotic Dynamics.* Cambridge University Press.

Bertels, K., Neuberg, L., Vassiliadis, S., and Pechanek, G. (1995) *XOR and Backpropagation Learning: In and Out of the Chaos?* ESANN 95 Conference Proceedings. Brussels, Belgium, pp. 69-74

Bertels, K., Neuberg, L., Vassiliadis, S., and Pechanek, G. (1998) *Chaos and Neural Network Learning.* Neural Processing Letters **7**, No. 2, pp. 69-80.

Biehl, M., and Schwarze, H. (1995) *Learning by On-Line Gradient Descent*. Journal of Physics, A28, pp. 643-656.

Boole, G. (1854) *An Investigation of The Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities.* Walton and Maberley, London.

Butler, E. (1993) *Bifurcations in Neural Learning*. 1993 Postgraduate Conference, Psychology Department, University of Nottingham.

Chang, W., and Mak, M. (1999) *A Conjugate Gradient Learning Algorithm for Recurrent Neural Networks*. Neurocomputing **24**, pp. 173-189.

Chomsky, N. (1957) *Syntactic Structures.* Mouton, The Hague.

Christensen, R. (1981) *Entropy Minimax Sourcebook.* Vol. 1. Entropy Limited, Lincoln, MA.

Craik, K.J.W. (1943) *The Nature of Explanation*. Cambridge University Press.

Daintith, J., and Nelson, R.D. (1995) *Dictionary of Mathematics.* Penguin Books.

Dane, F. (1990) *Research Methods.* Brooks/Cole Publishing Company.

Ditto, W.L., Spano, M.L., and Lindner, J.F. (1995) *Techniques for the Control of Chaos.* Physica D **86,** pp. 198-211.

Fahlman, S.E. (1988) *An Empirical Study of Learning Speed in Back-Propagation Networks.* Technical Report CMU-CS-88-162, Carnegie-Mellon University, Computer Science Dept., Pittsburgh, PA.

Fahlman, S.E., and Lebiere, C. (1989) *The Cascade-Correlation Learning Architecture.* In: Advances in Neural Information Processing Systems II, (Ed. Touretzky, D.S.), Morgan Kaufmann, San Mateo, pp. 542-532.

Feigenbaum, M. (1980) *Universal Behaviour in Non-linear Systems.* Los Alamos Science/Summer 1980, pp. 4-27.

Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., and Lee, Y.C. (1992) *Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks.* Neural Computation, Vol. 4, no. 3, pp. 393-405.

Gleick, J. (1987) *Chaos*. Penguin Books, New York.

Gorse, D., Shepherd, A., and Taylor, J.G. (1993) *Avoiding Local Minima using a Range Expansion Algorithm.* Neural Network World, 5, pp. 503-510.

Herz, J., Krogh, A., and Palmer, R. (1991) *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood, CA.

Hinton, G., and Sejnowski, T. (1986). *Learning and Re-Learning in Boltzmann Machines.* In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. (Eds. Rumelhart, D.E. and McClelland, J.L.). MIT Press, Cambridge, 1986.

Hochreiter, S., and Schmidhuber, J. (1997) *Flat Minima*. Neural Computation 9, pp. 1–42.

Hofstadter, D.R. (1980) *Godel, Escher, Bach: An Eternal Golden Braid*. Penguin Books, London.

Hopfield, J.J. (1982) *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*. Proceedings US National Academy of Science 79, pp. 2554-2558.

Hopfield, J.J., and Tank, D.W. (1987) *Collective Computation in Neuronlike Circuits.* Scientific American, December 1987.

Howes, P., and Crook, N. (1999) *Using Input Parameter Influences to Support the Decisions of Feedforward Neural Networks*. Neurocomputing **24**, pp. 191-206.

Hung, E.S. (1995) *Parameter Estimation in Chaotic Systems* M.I.T A.I. Technical Report No. 1541

Hunt, E., and Johnson, G. (1993) *Keeping Chaos at Bay.* IEEE spectrum, November 1993, pp. 32-36.

Kolen, J.F., and Pollack, J.B. (1990) *Back Propagation is Sensitive to Initial Conditions.* Complex Systems, **4**, pp. 269-280.

Kramer, A., and Vincentelli, S. (1989) *Efficient Parallel Learning Algorithms for Neural Networks.* In: Advances in Neural Information Processing I (Ed. Touretzky, D.S.), Morgan Kaufman, San Mateo.

Lauwerier, H.A. (1991) *Fractals: Images of Chaos*, Princeton University Press, Oxford.

Le Cun, Y. (1985) *Proc. Cognitiva* **85**, pp. 599-604.

Liang, P. (1991) *Designing Artificial Neural Networks based on the Principle of Divide-and-Conquer.* In: Proceedings of International Conference on Circuits and Systems, pp. 1319-1322.

Lisboa, P.J.G., and Perantonis, S.J. (1991) *Complete Solution of the Local Minima in the XOR Problem.* Network **2**, pp. 119-124.

MacKay, D.J.C. (1991) *Maximum Entropy Connections: Neural Networks*. In: Maximum Entropy and Bayesian Methods, (Eds. Grandy, W. and Schick, L.), Kluwer Academic Publishers, pp. 237-244.

Mak, M.W., Ku, K.W., and Lu, Y.L. (1999) *On the Improvement of the Real Time Recurrent Learning Algorithm for Recurrent Neural Networks.* Neurocomputing **24**, pp. 13-36.

Manausa M.E., and Lacher, R.C. (1991*) Chaos and the Stepsize Dilemma in the Backpropagation Learning Algorithm*. Proceedings WNN 91, pp. 153-160.

Manausa, M.E., and Lacher, R.C. (1993) *Parameter Sensitivity in the Backpropagation Learning Algorithm*. IEEE Spectrum, December 1993.

Marsaglia, G. (1993) *Remarks on Choosing and Implementing Random Number Generators.* Communications of the ACM, Vol. 36, No.7, pp. 105-108.

McCulloch, W.S., and Pitts, W. (1943) *A Logical Calculus of the Ideas Immanent in Nervous Activity.* Bulletin of Mathematical Biophysics **5**, pp. 115-133.

Melnik, O., and Pollack, J.B. (1998) *A Gradient Descent Method for a Neural Fractal Memory.* WCCI 98 (IJCNN), IEEE.

Minsky, M.L. and Papert, S.A. (1969) *Perceptrons*. MIT Press, Cambridge, MA.

Minsky, M.L. and Papert, S.A. (1988) *Perceptrons: An Introduction to Computational Geometry, Expanded Edition*. MIT Press, Cambridge, MA.

Mulholland, H. (1976) *Calculus*. W. H. Allen, London.

Murray, A.F. (1991) *Analog VLSI and Multi-Layer Perceptrons - Accuracy, Noise and On-Chip Learning.* In: Proceedings of Second International Conference on Microelectronics for Neural Networks, pp. 27-34.

Parker, D.B. (1985) *Learning-Logic.* Technical Report TR-47, Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, Cambridge, MA, 1985.

Pinkas, G. and Dechter, R. (1995) *Improving Connectioninst Energy Minimization*. Journal of Artificial Intelligence Research 3 (1995), AIAF / Morgan Kaufmann, pp. 223-248.

Pollack, J.B. (1989a) *No Harm Intended: A Review of the Perceptrons Expanded Edition*. Journal of Mathematical Psychology, 33, 3, pp. 358-365.

Pollack, J.B. (1989b) *Implications of Recursive Auto Associative Memories*. In: Advances in Neural Information Processing Systems II, (Ed. Touretzky, D.S.), Morgan Kaufmann, San Mateo, pp. 527-536.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992) *Numerical Recipes in C.* Cambridge University Press, Cambridge, UK.

Qian, N. (1999) *On the Momentum Term in Gradient Descent Learning Algorithms*. Neural Networks **12**, pp. 145-151.

Reichgelt, H. (1990) *Knowledge Representation: an A.I. Perspective*. Ablex Publishing Corp.

Renals, S. (1990) *Chaos in Neural Networks.* In: Proceedings of EURASIP Neural Networks Workshop (Eds. Aleida, L. and Wellekens, C.). Sesimbra, Portugal, 1990, pp. 90-99.

Riedmiller, M., and Braun, H. (1993) *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm.* In: Proceedings of the IEEE International Conference on Neural Networks (Ed. Ruspini, H.). San Francisco, 1993, pp. 586-591.

Ritter, F. E. (1991) *Towards Fair Comparisons of Connectionist Algorithms through Automatically Generated Parameter sets.* In: Proceedings of the 13th Annual Conference of the Cognitive Science Society. Lawrence Erlbaum, Hillsdale, NJ, pp. 877-881.

Rosenblatt, F. (1961) *Principles of Neurodynamics*. Spartan, Washington, DC.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986a) *Learning Internal Representations by Error Propagation*. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. (Eds. Rumelhart, D.E. and McClelland, J.L.). Ch. 8. MIT Press, Cambridge, 1986.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986b) *Learning Representations by Back-Propagating Errors*. Letters to nature, Vol. 323 / October 1986.

Rumelhart, D.E. and McClelland, J.L. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol. 1. MIT Press, Cambridge, 1986.

Rumelhart, D.E. and McClelland, J.L. (1988) *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* MIT Press.

Shannon, C.E. (1938) *A Symbolic Analysis of Relay and Switching Circuits.* Transactions American Institute of Electrical Engineers, Vol. 57, pp. 713-723.

Shannon, C.E. (1949) *The Mathematical Theory of Communication.* In: *The Mathematical Theory of Communication* (With Warren Weaver) University of Illinois Press, Urbana.

Smolensky, P. (1986) *Information Processing in Dynamical Systems: Foundations of Harmony Theory.* In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. (Eds. Rumelhart, D. E. and McClelland, J. L.). MIT Press, Cambridge, 1986.

Smoot, G., and Davidson, K. (1993) *Wrinkles in Time: The Imprint of Creation*. Abacus Books, London.

Someya, K., Shinozaki, H., and Sekine, Y. (1999) *Pulse-Type Hardware Chaotic Neuron Model and its Bifurcation Phenomena.* Neural Networks 12, pp. 153–161.

Sprinkhuizen-Kuyper, I.G. and Boers, E.J.W. (1996a) *The Error Surface of the Simplest XOR Network has only Global Minima*. Neural Computation **8**, pp.1301-1320.

Sprinkhuizen-Kuyper, I.G. and Boers, E.J.W. (1996b) *The Error Surface of the 2-2-1 XOR Network: Stationary Points with Infinite Weights.* Technical Report 96-10, Department of Computer Science, Leiden University.

Tsung, F.S. and Cottrell, G.W. (1993) *Phase-Space Learning for Recurrent Networks.* Technical Report CS93-285, Dept. Computer Science and Engineering, University of California, San-Diego.

Turega, M. (1992) *A Computer Architecture to Support Neural Net Simulation.* The Computer Journal, Vol. 35, No 4., 1992, pp. 353-360.

Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T. and Alkon, D.L. (1988*) Accelerating the Convergence of the Back-Propagation Method.* Biological Cybernetics **59**, pp. 257-263.

Wang, S. and Hsu, C. (1995*) Terminal Attractor Learning Algorithms for Back Propagation Neural Networks*. In: Proceedings of IEEE conference on Neural Networks, Taiwan, 1995, pp. 183-189.

Werbos, P. J. (1974*) Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences.* PhD thesis, Harvard University, 1974.

Widrow, G. and Hoff, M.E. (1960) *Adaptive Switching Circuits*. Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, pp. 96-104.

Wightman, W.P.D. (1951) *The Growth of Scientific Ideas.* Yale University Press, New Haven.

Williams, R.J. and Zipser, D. (1994) *Gradient-Based Learning Algorithms for Recurrent Networks and their Computational Complexity.* In: Backpropagation: Theory, Architectures, and Applications (Eds. Chauvin, Y. and Rumelhart, D.E.), Ch. 13. Lawrence Erlbaum, Hillsdale, NJ, pp. 433-486.

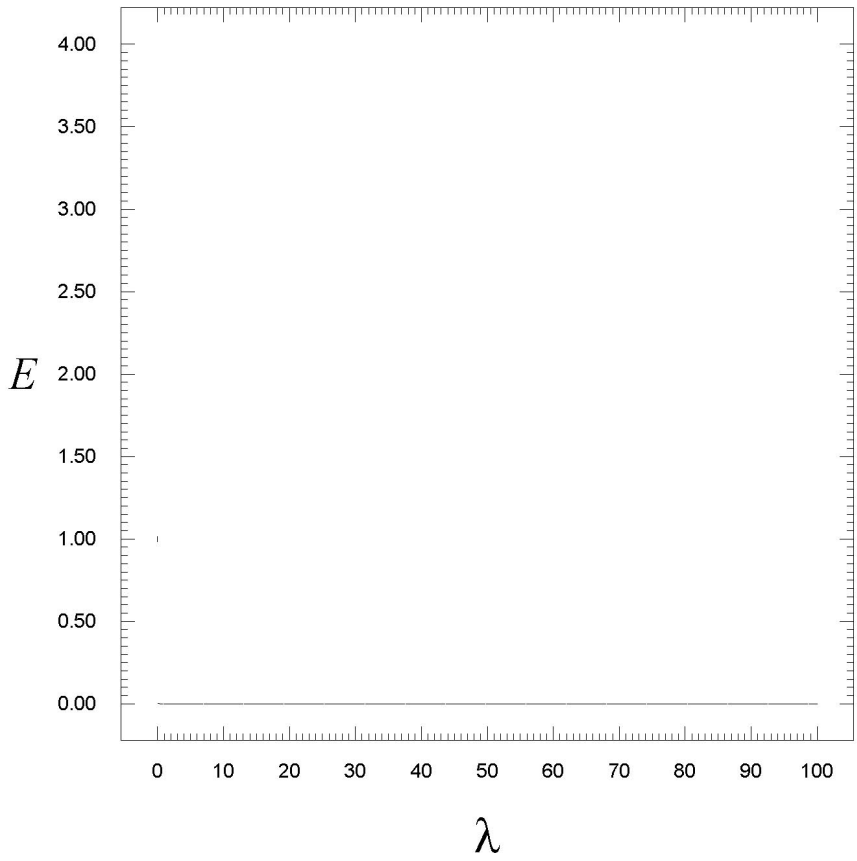
Woods, D. (1988) *Back and Counter Propagation Aberrations.* In: Proceedings of International Joint Conference on Neural Networks, Vol. 1, pp. 343-353.

Yorke, J.A. and Li, T.Y., (1975) *Amer. Math. Monthly* **82,** pp. 985.

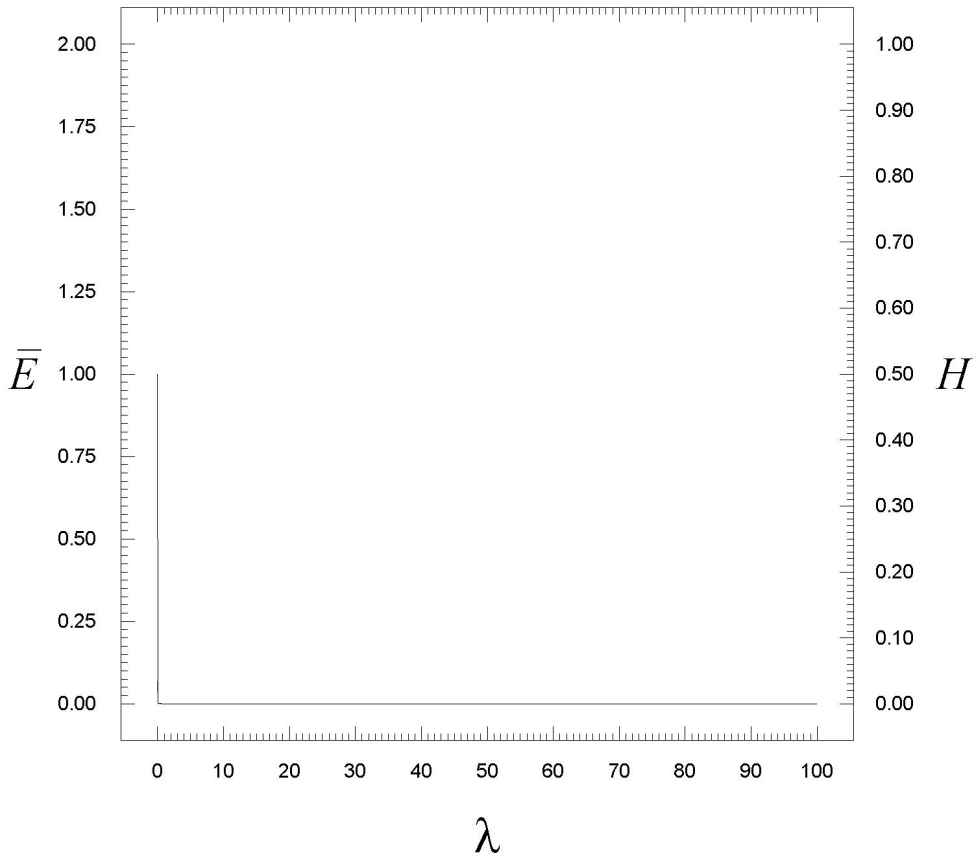
Appendix A

This appendix contains all of the individual dynamical mappings from the results of testing the 2:1:1 biased feedforward network on all of the binary Boolean functions at 1,000 epochs. The results are presented from testing each individual function are listed in order of function number, first by presenting a bifurcation diagram of the sum squared error as a function of the lambda parameter, followed by a mean sum squared error and smoothed entropy diagram.

Function 0: f(A,B) = FALSE

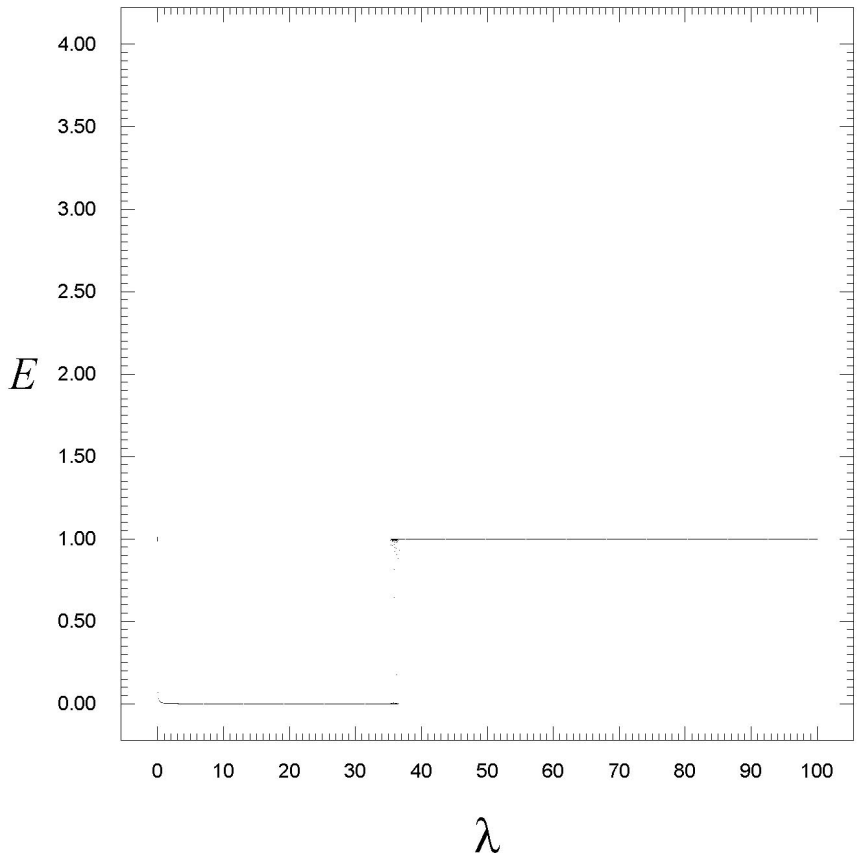


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 0 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.

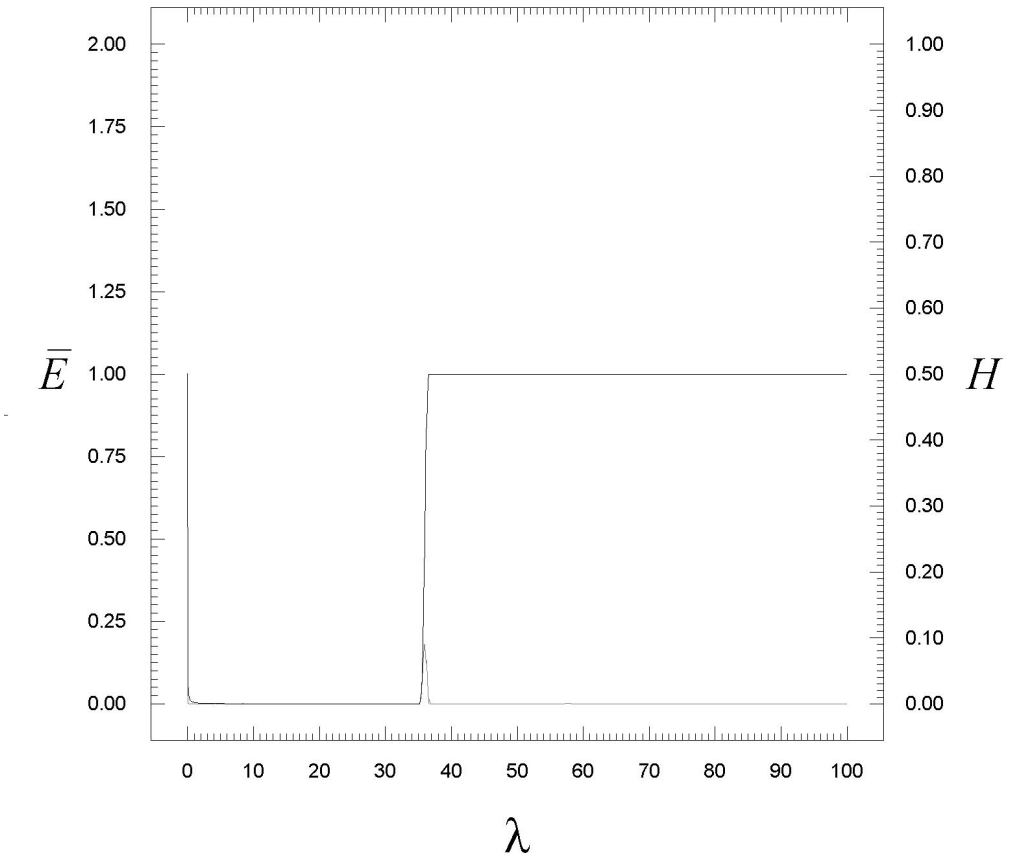


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 0 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [0 < E < 4]. The maximum of E is 0.999646 , @ lambda= 0.The maximum of H is 0.068952 , @ lambda= 0.

Function 1: f(A,B) = A AND B

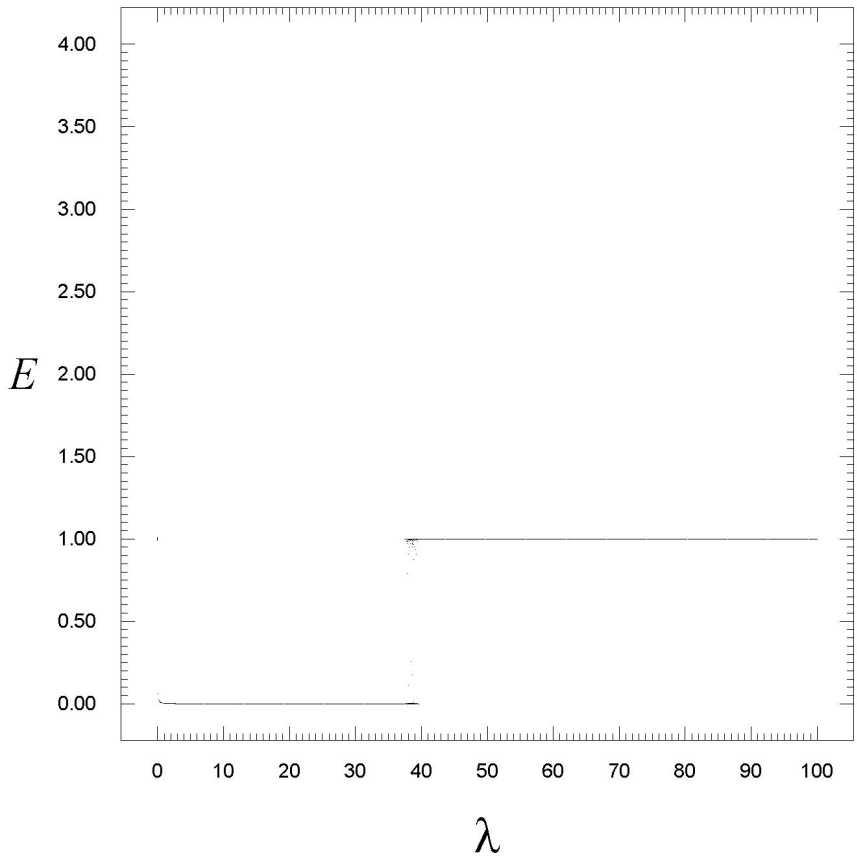


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 1 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 35.2

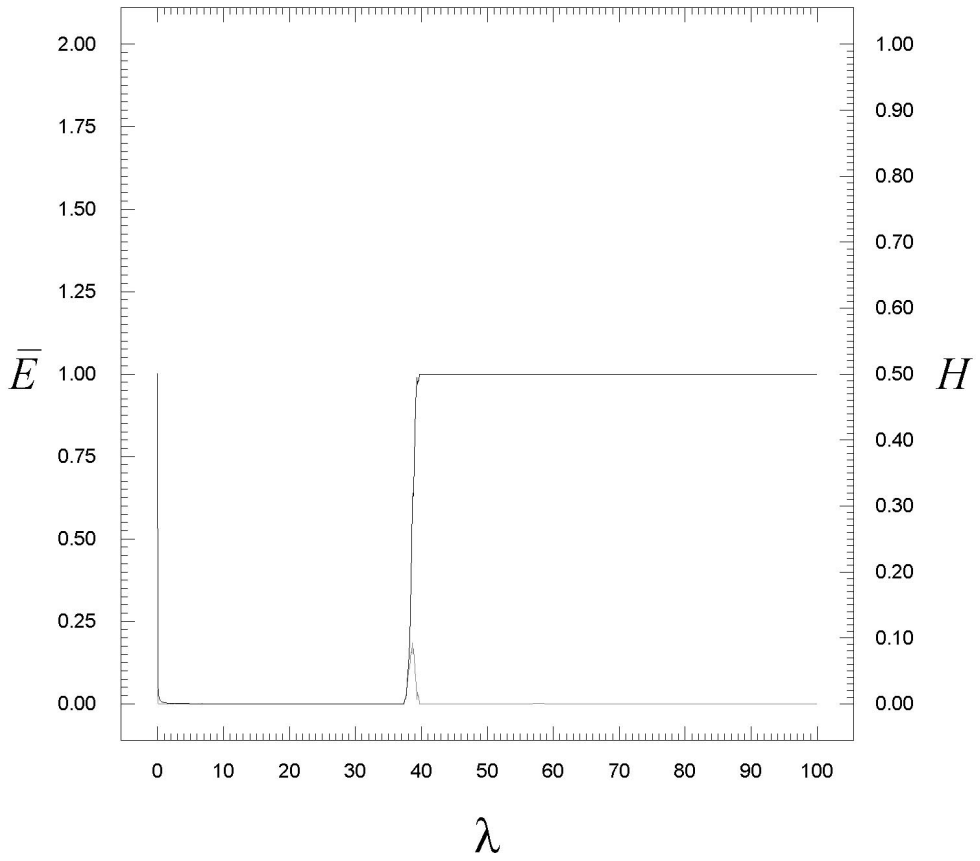


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 1 using standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000276 , @ lambda= 0. The maximum of H is 0.181502 , @ lambda= 35.9.

Function 2: f(A,B)=A AND ~B

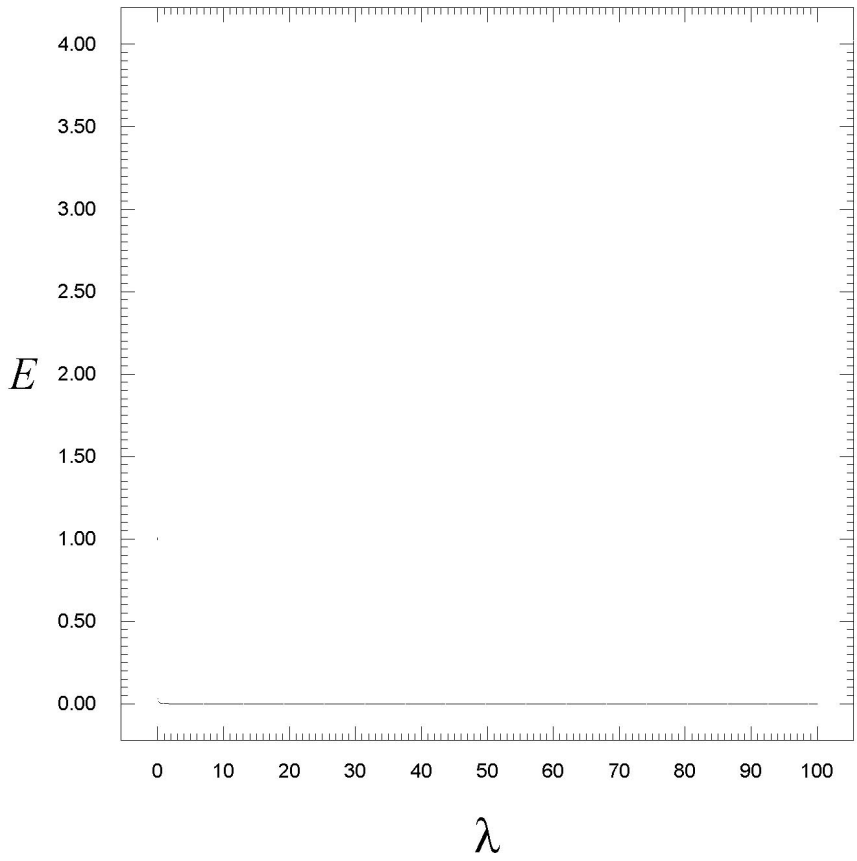


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 2 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 37.4.

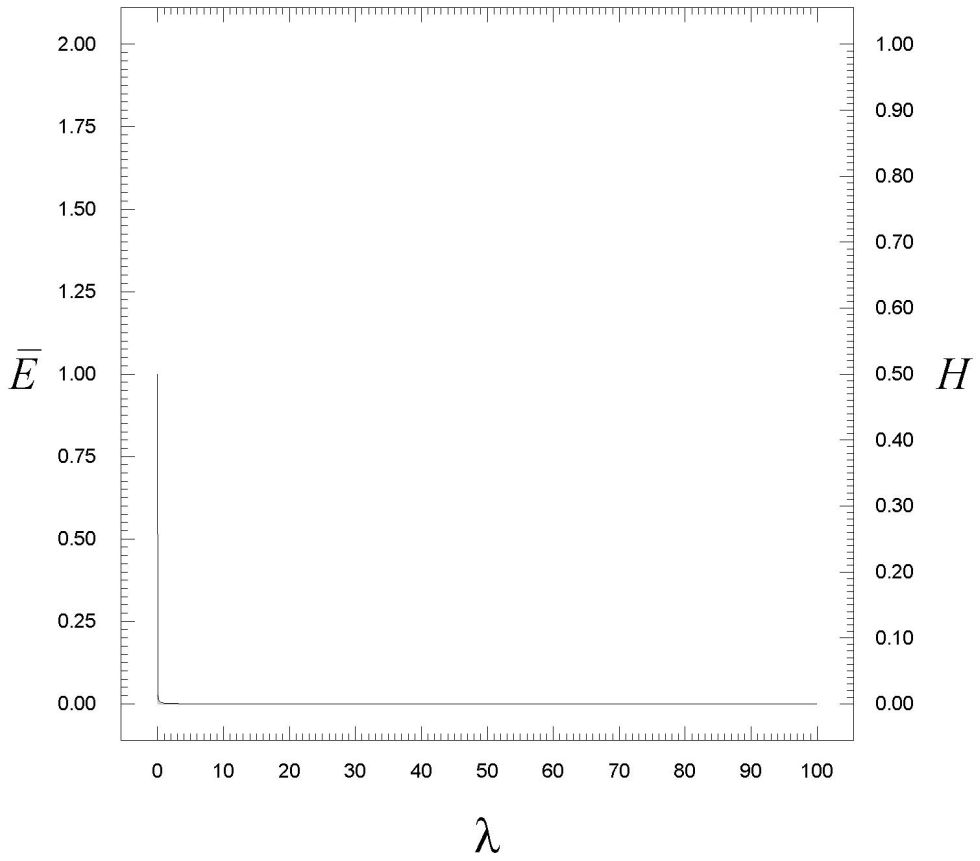


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 2 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000458 , @ lambda= 0. The maximum of H is 0.184454 , @ lambda= 38.6.

Function 3: f(A,B) = A

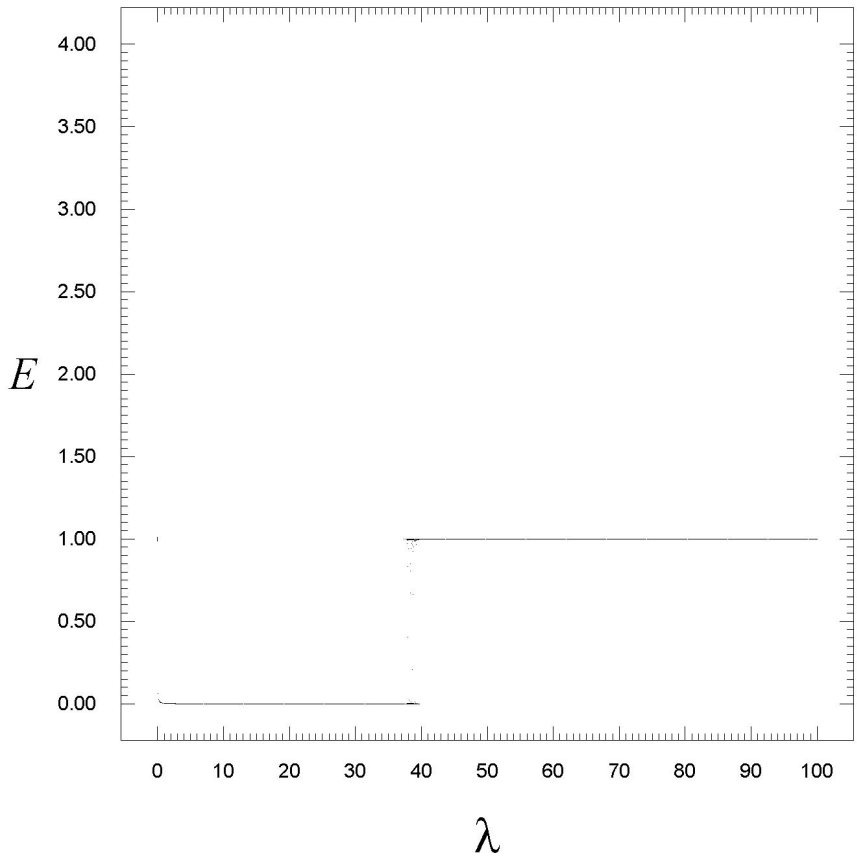


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 3 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.

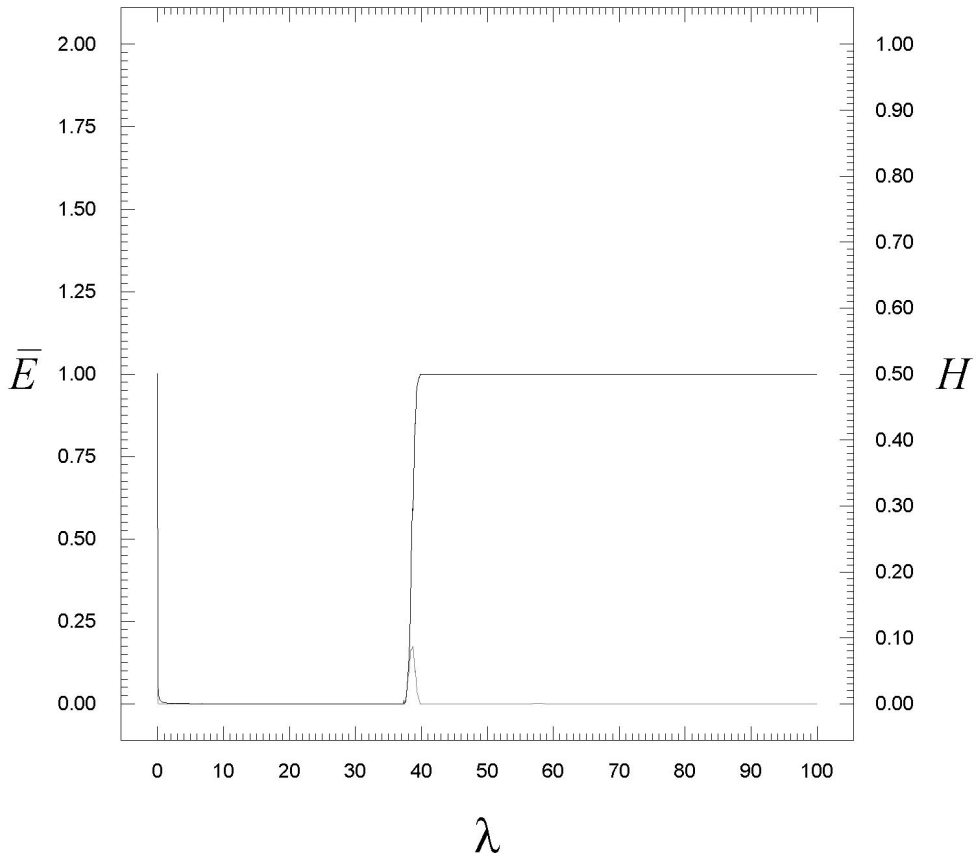


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 3 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 0.999681 , @ lambda= 0. The maximum of H is 0.025263 , @ lambda= 0.

Function 4: f(A,B) = ~A AND B

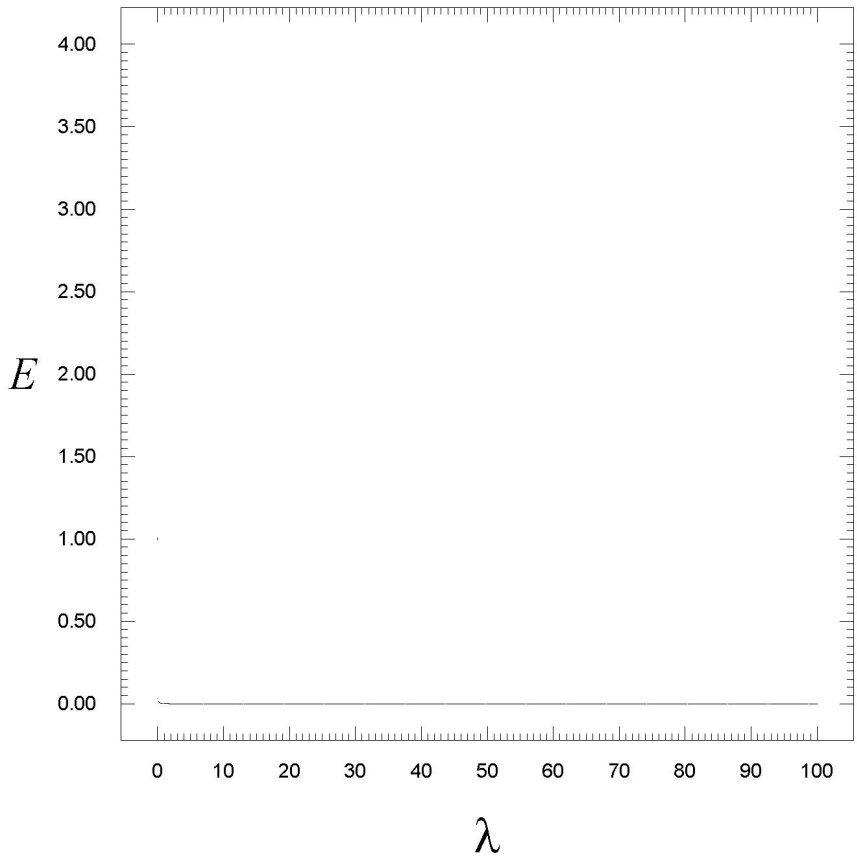


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 4 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 37.5.

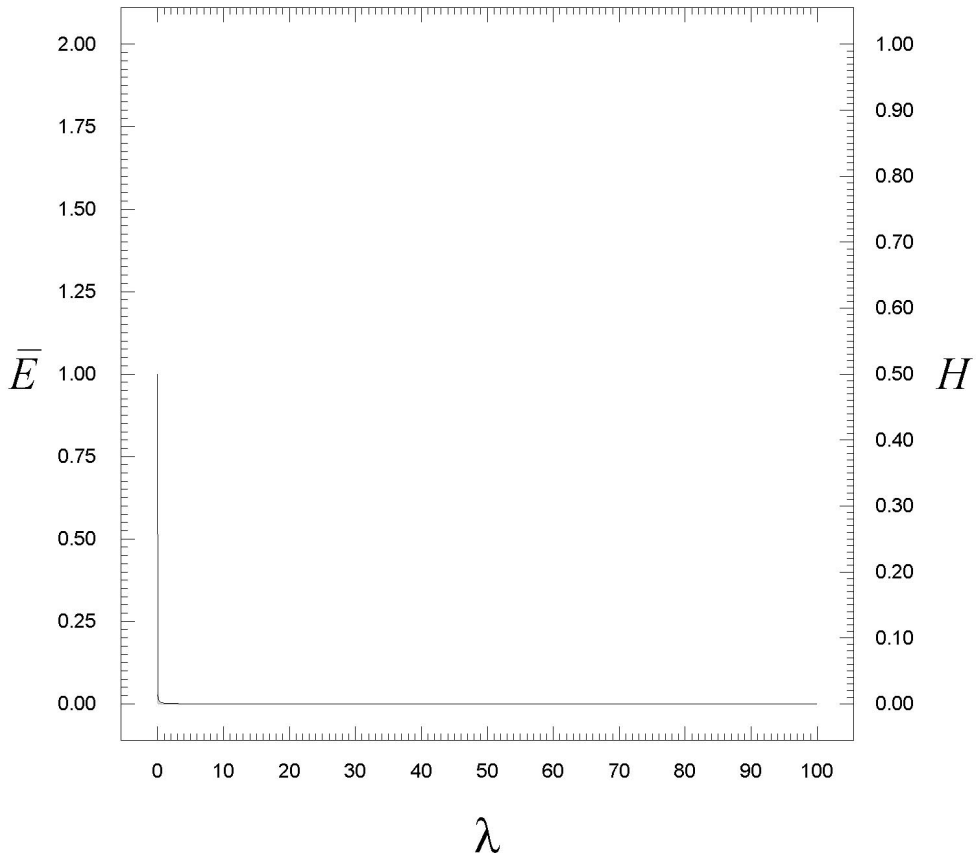


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 4 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000684 , @ lambda= 0. The maximum of H is 0.173404 , @ lambda= 38.6.

Function 5: f(A,B) = B

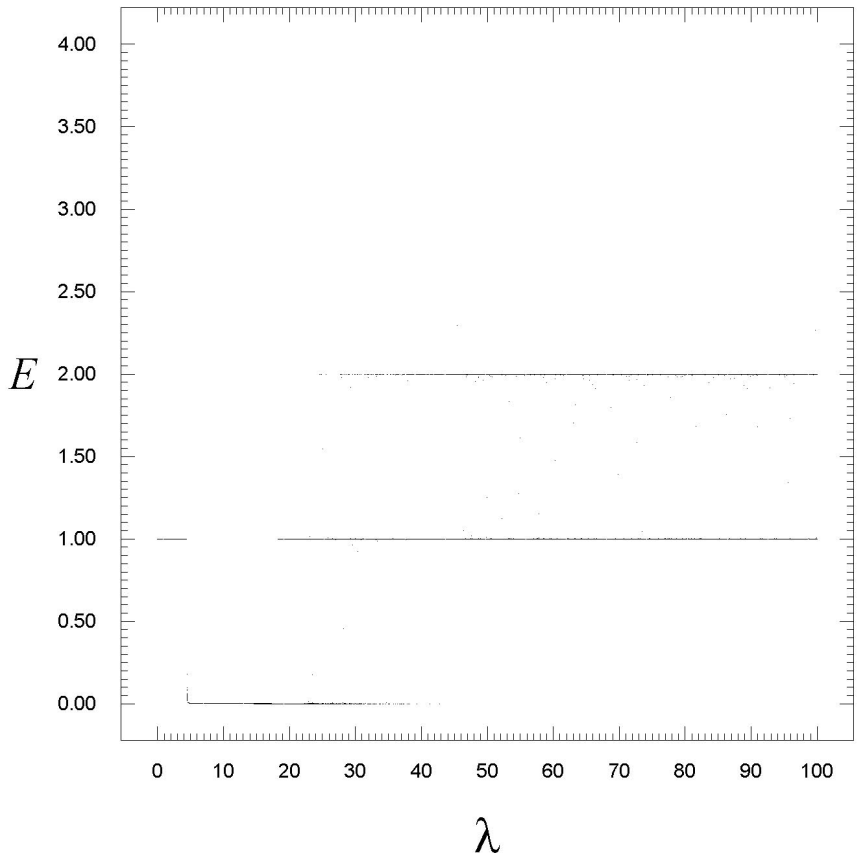


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 5 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.

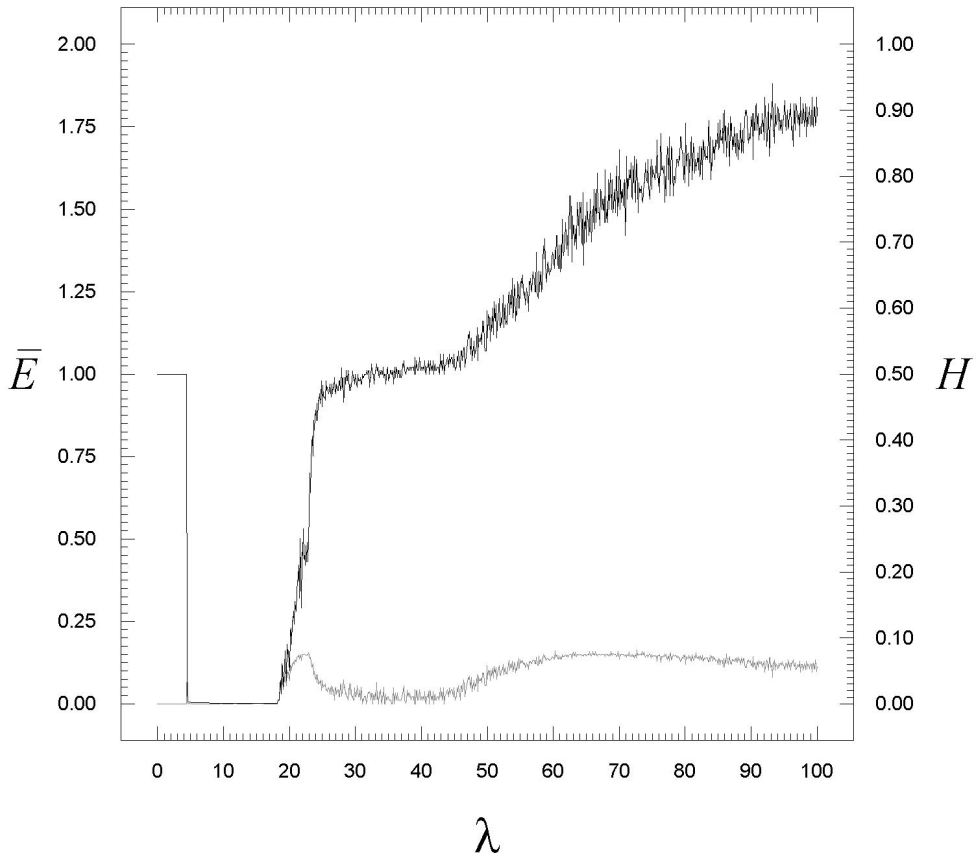


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 5 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [0 < E < 4]. The maximum of E is 1.000213 , @ lambda= 0. The maximum of H is 0.026872 , @ lambda= 0 .

Function 6: f(A,B) = A XOR B

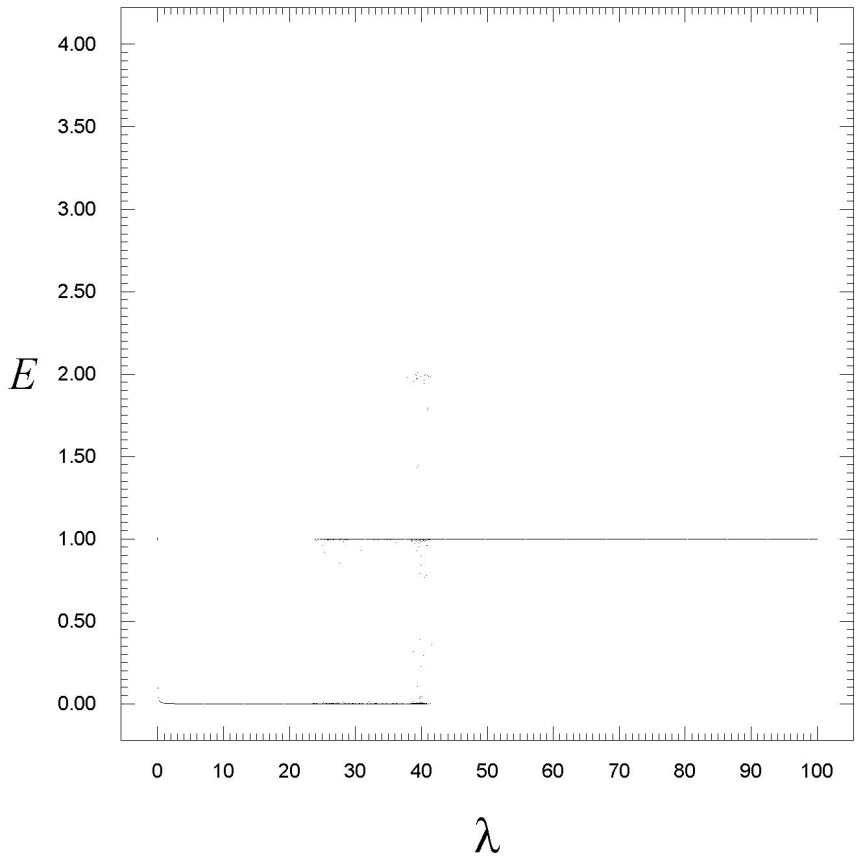


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 6 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 4.6 and ends @ lambda= 18.2.

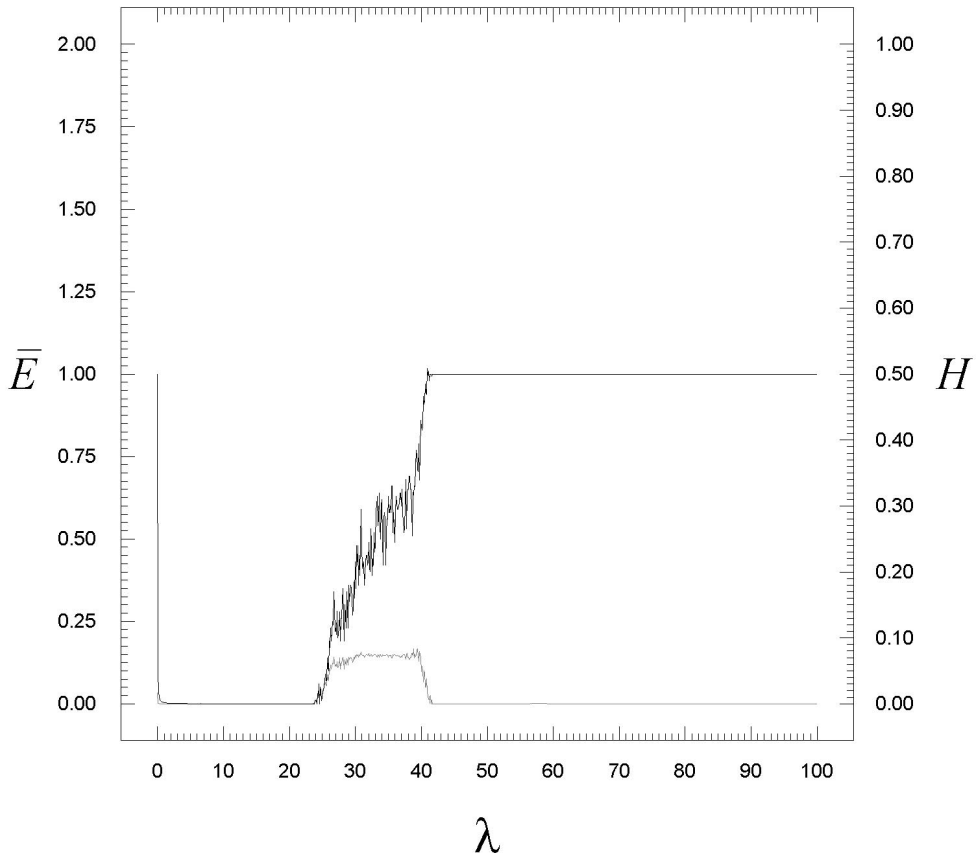


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 6 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.879992 , @ lambda= 93.2. The maximum of H is 0.162057 , @ lambda= 72.7 .

Function 7: f(A,B) = A OR B

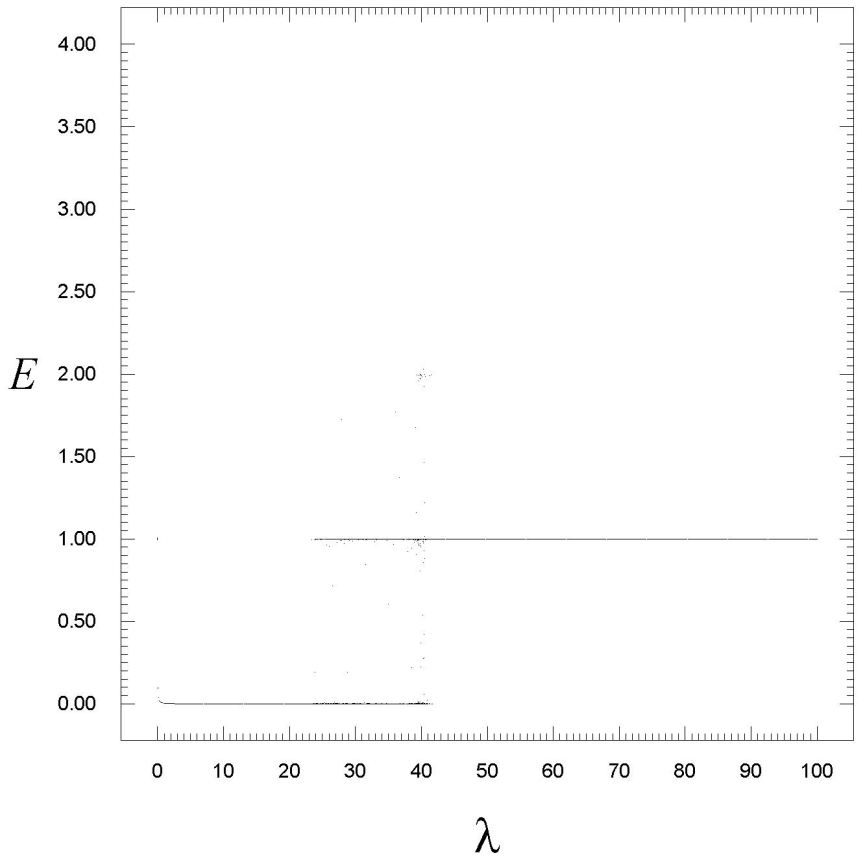


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 7 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 24.6.

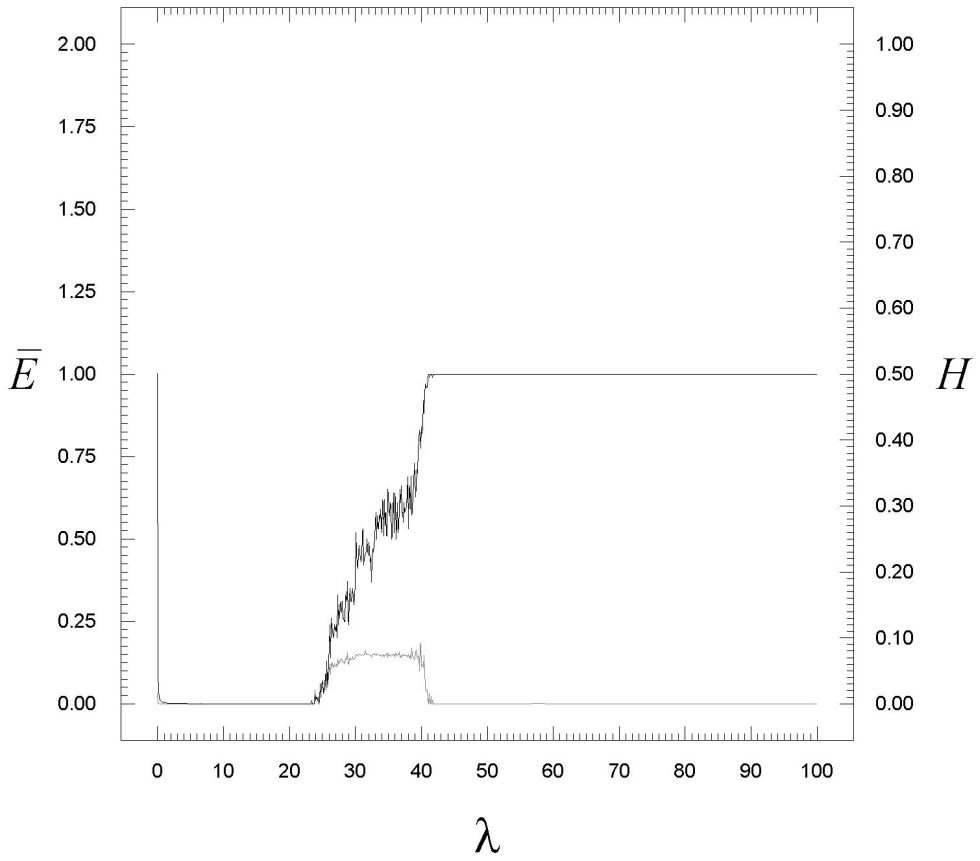


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 7 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.017339 , @ lambda= 40.9. The maximum of H is 0.167025 , @ lambda= 39.4.

Function 8: f(A,B) = A NOR B

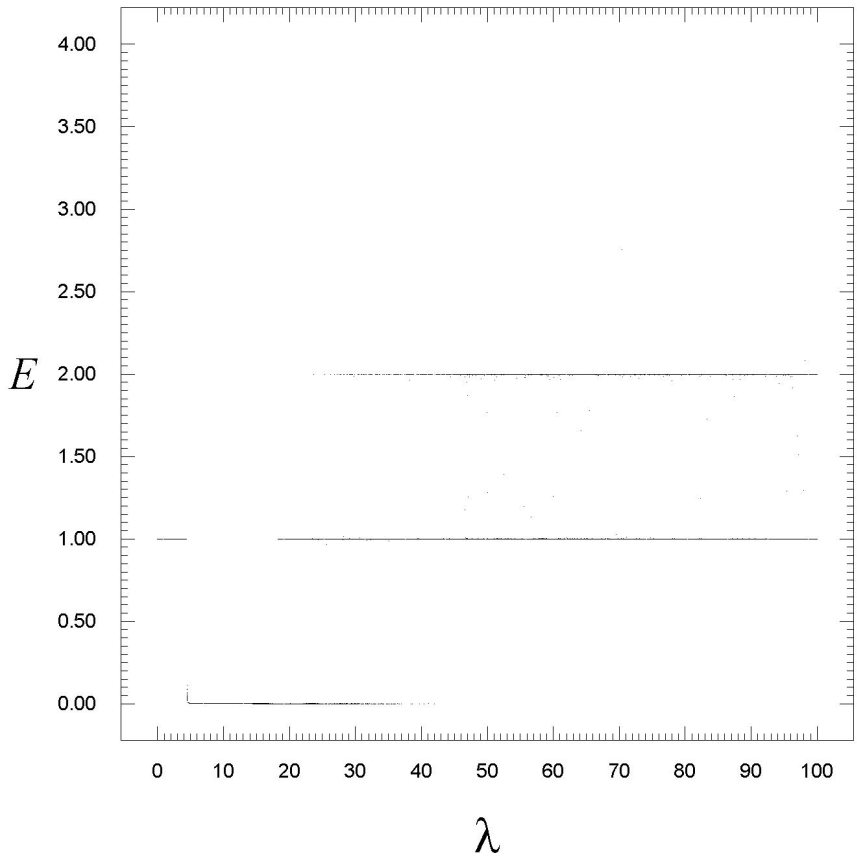


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 8 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 24.5.

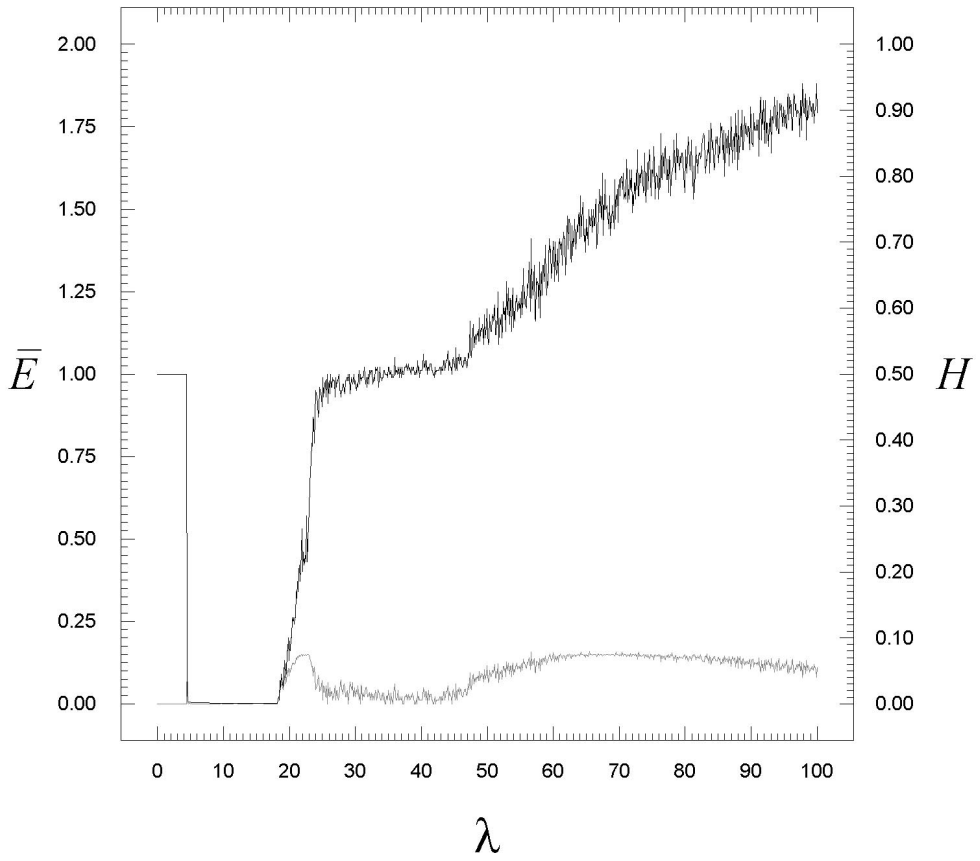


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 8 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000329 , @ lambda= 0. The maximum of H is 0.184027 , @ lambda= 39.9.

Function 9: f(A,B) = A XNOR B

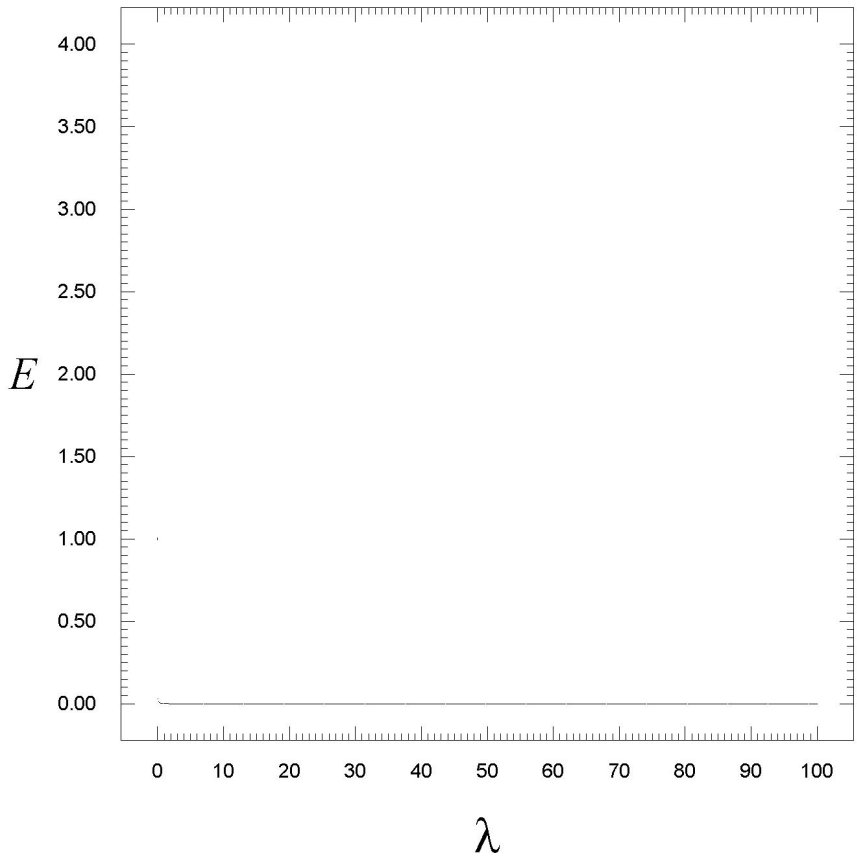


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 9 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 4.6 and ends @ lambda= 18.2.

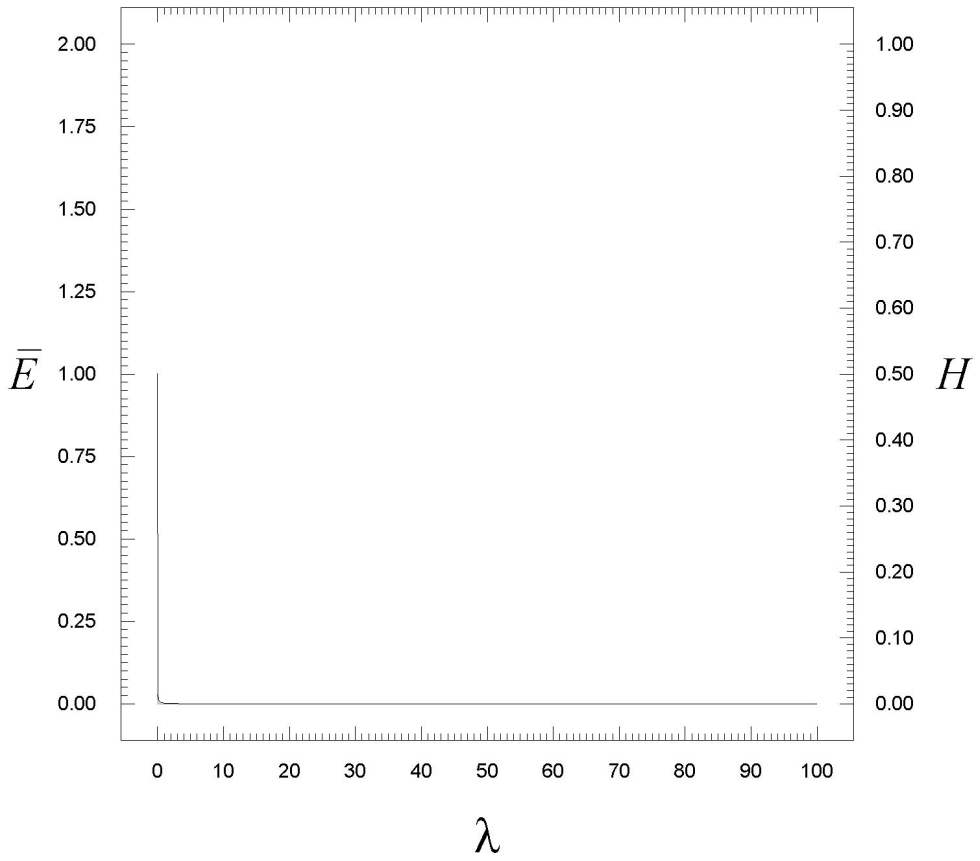


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 9 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.87999 , @ lambda= 97.8. The maximum of H is 0.159267 , @ lambda= 65.5.

Function 10: f(A,B) = ~B

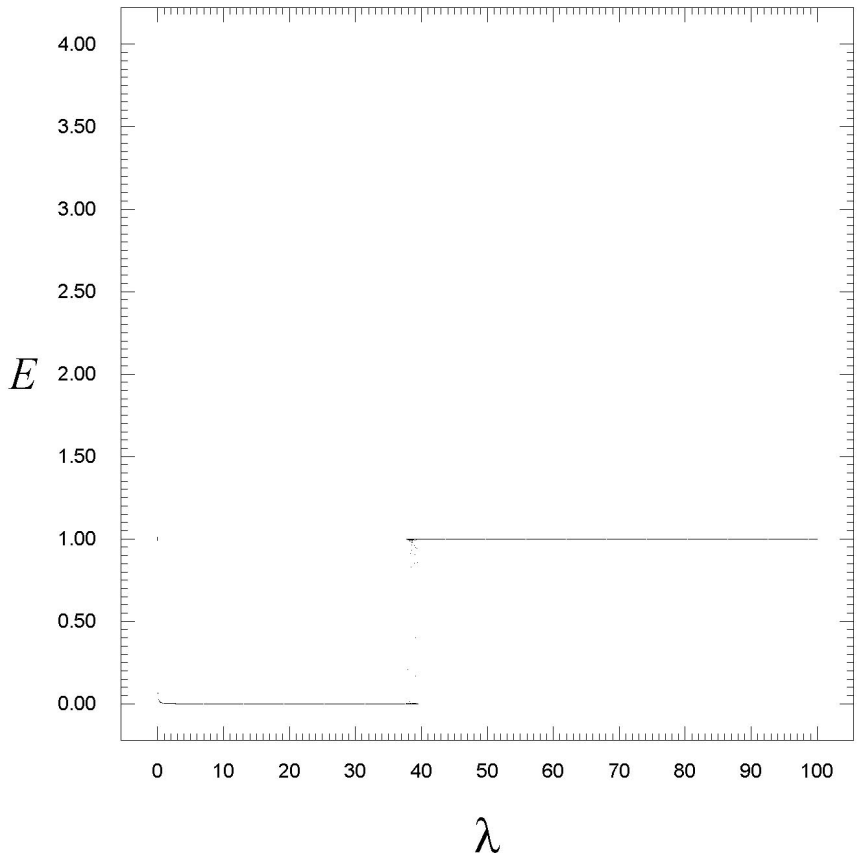


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 10 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.

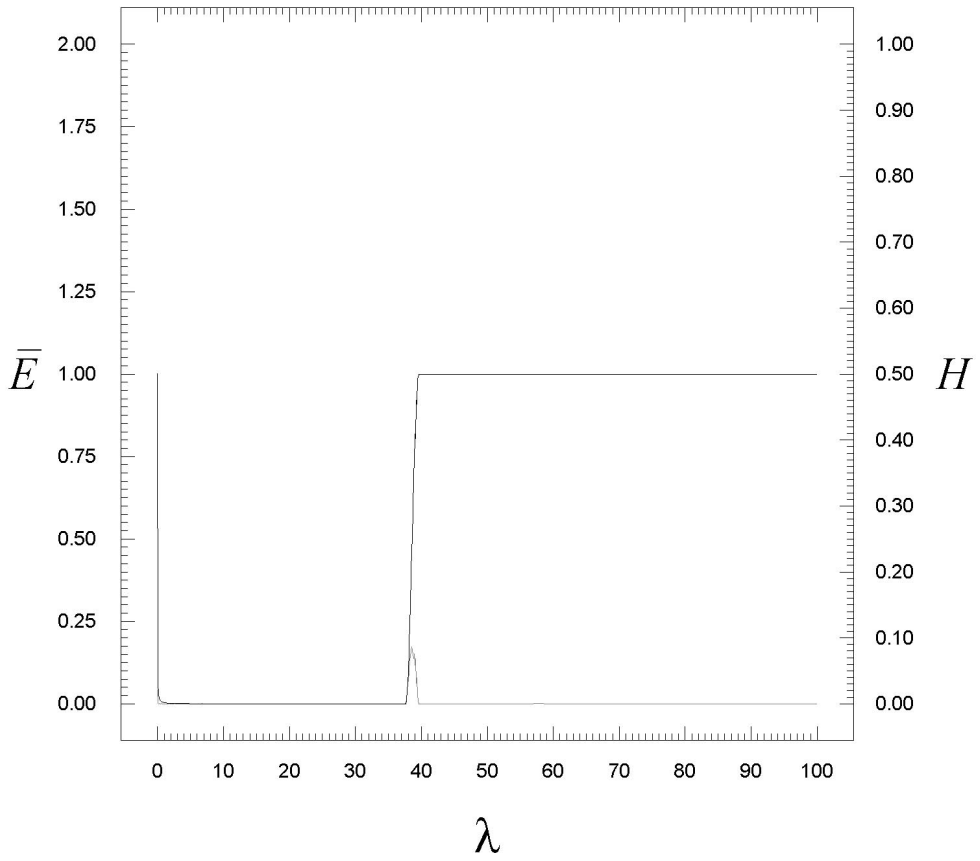


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 10 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000332 , @ lambda= 0. The maximum of H is 0.02779 , @ lambda= 0.

Function 11: f(A,B) = A OR ~B

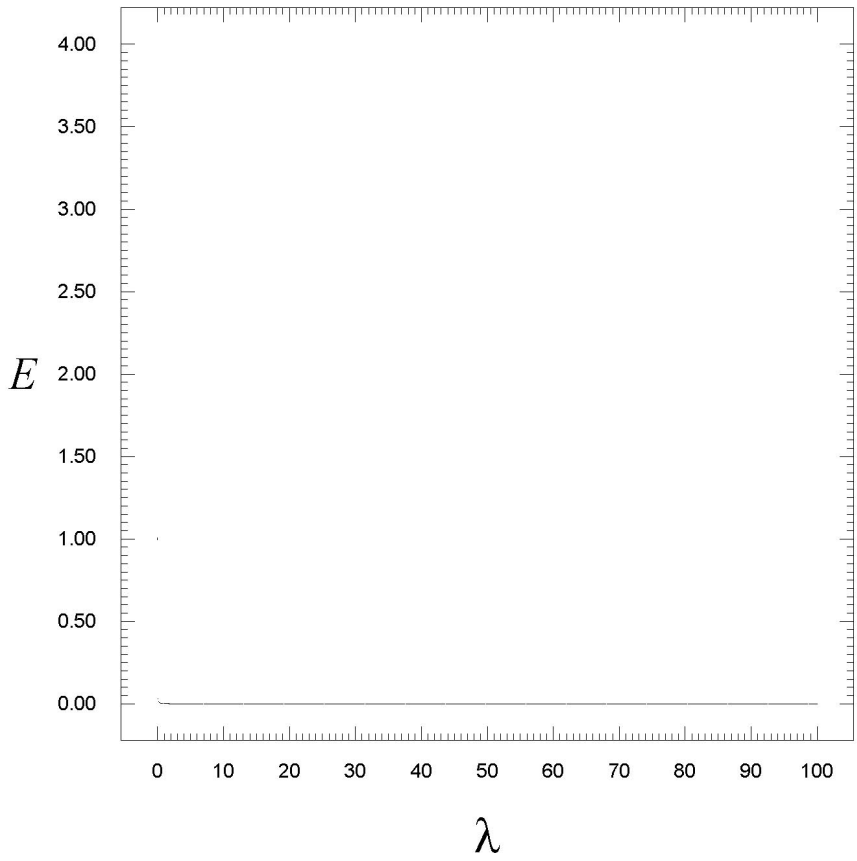


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 11 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 37.6.

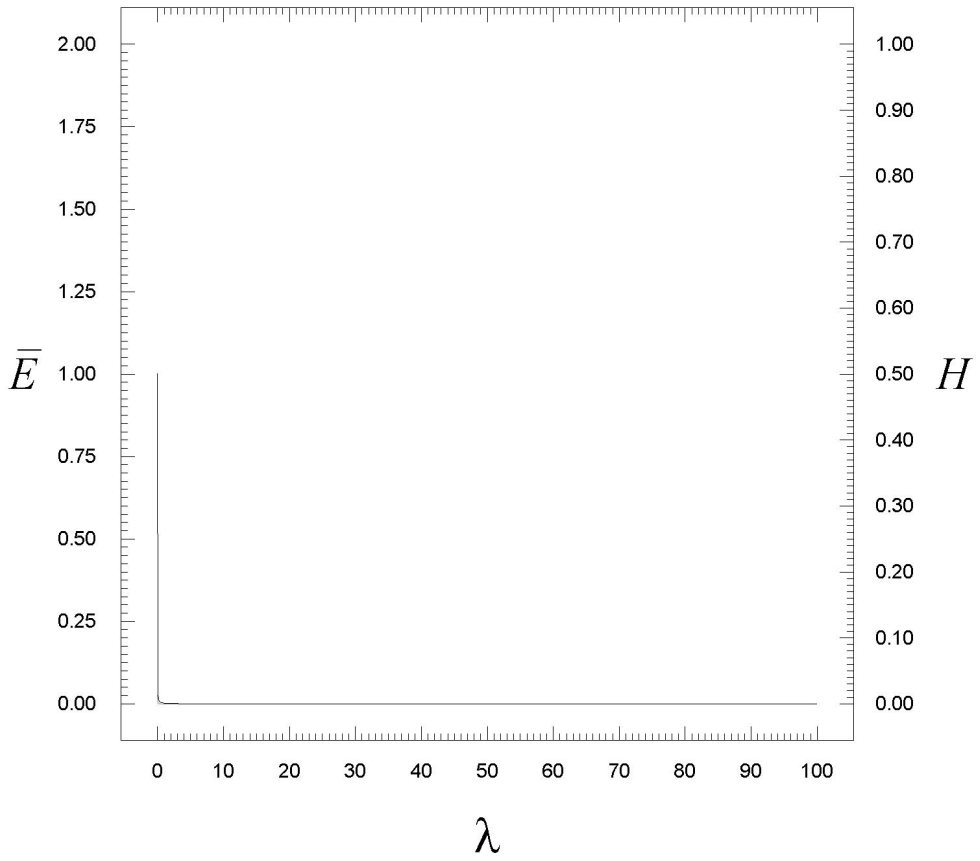


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 11 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [0 < E < 4]. The maximum of E is 1.000493 , @ lambda= 0. The maximum of H is 0.173311 , @ lambda= 38.5.

Function 12: f(A,B) = ~A

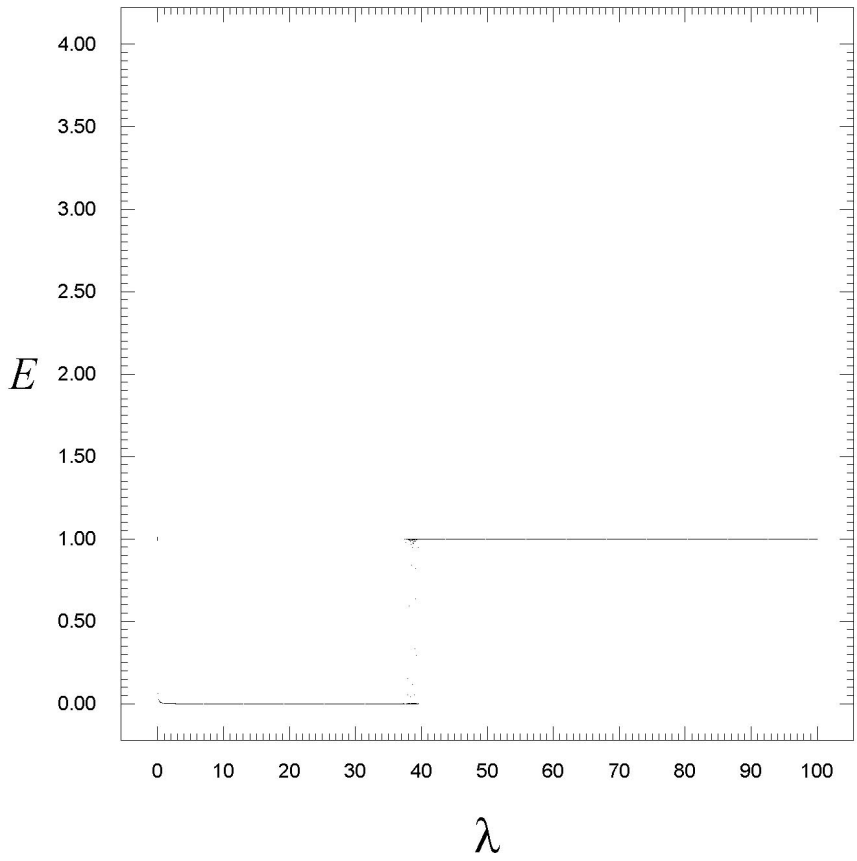


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 12 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.

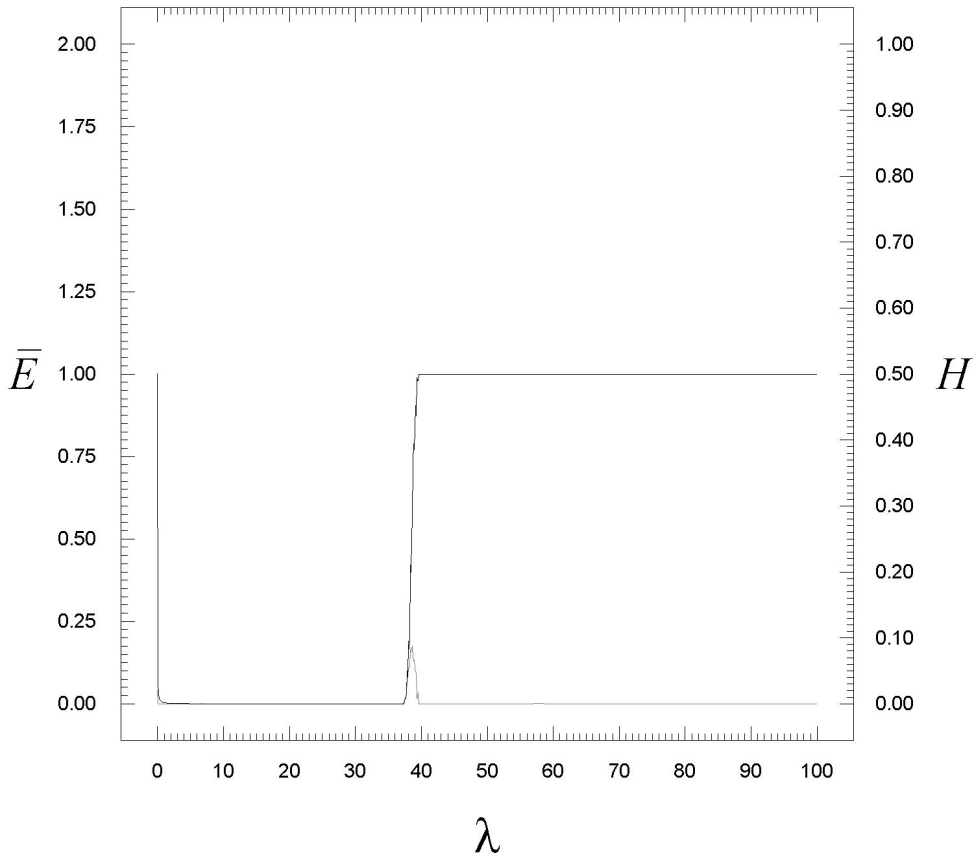


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 12 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000768 , @ lambda= 0. The maximum of H is 0.025197 , @ lambda= 0.

Function 13: f(A,B) = ~A OR B

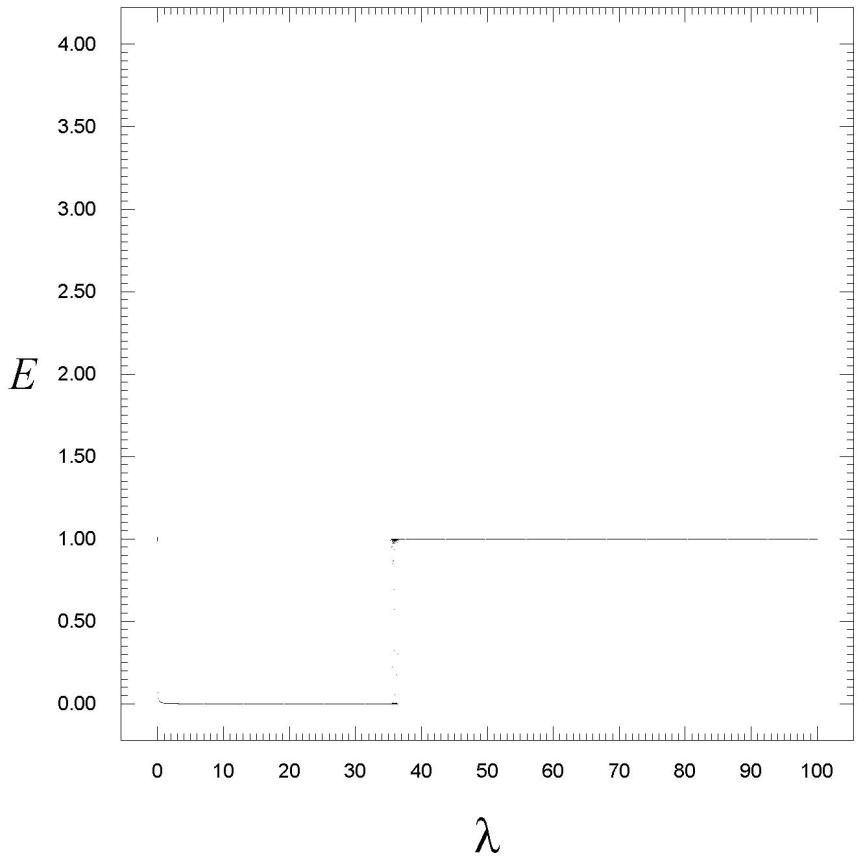


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 13 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 37.4.

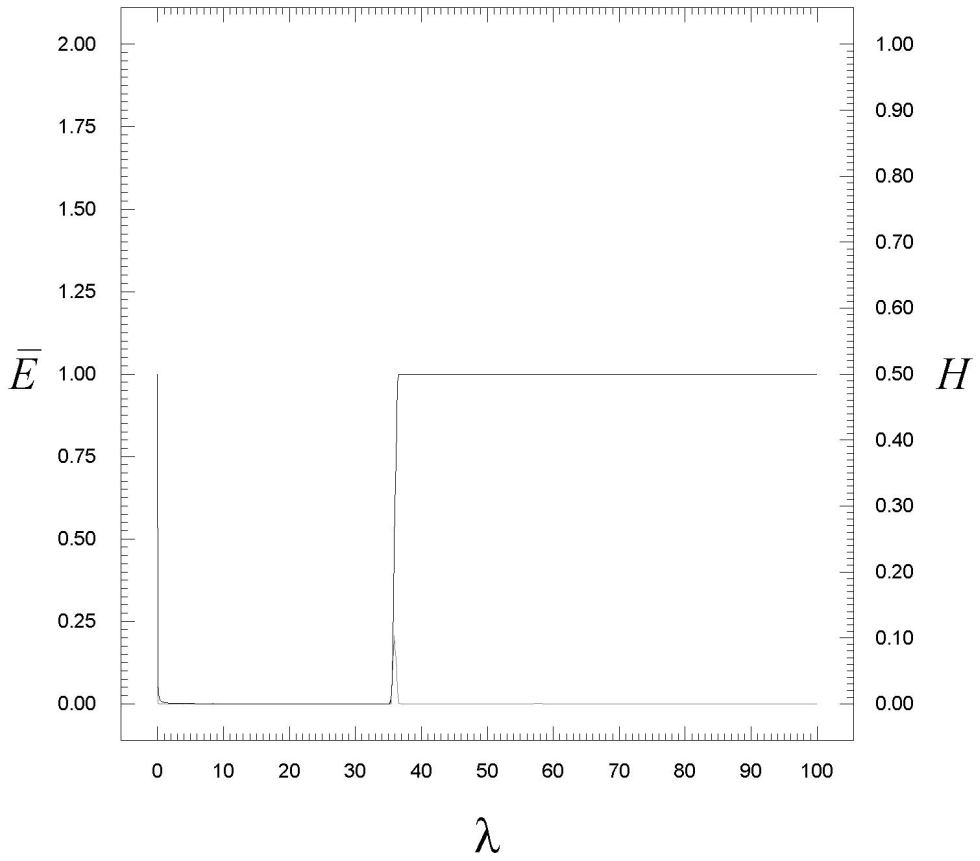


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 13 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000817 , @ lambda= 0. The maximum of H is 0.176311 , @ lambda= 38.6.

Function 14: f(A,B) = A NAND B

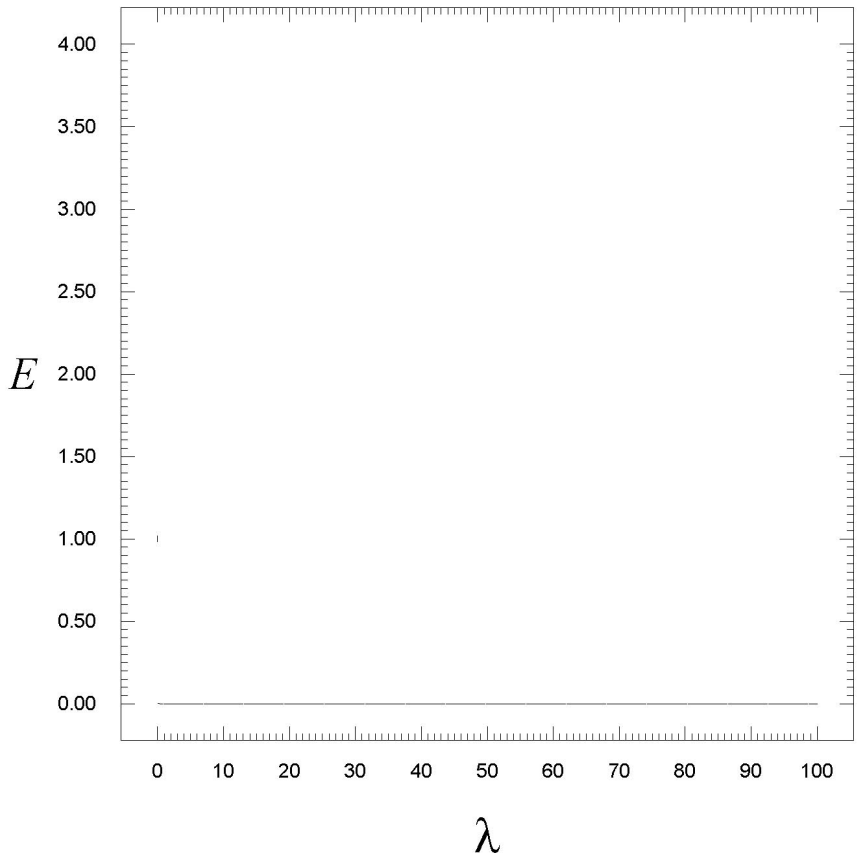


Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 14 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.2 and ends @ lambda= 35.4.

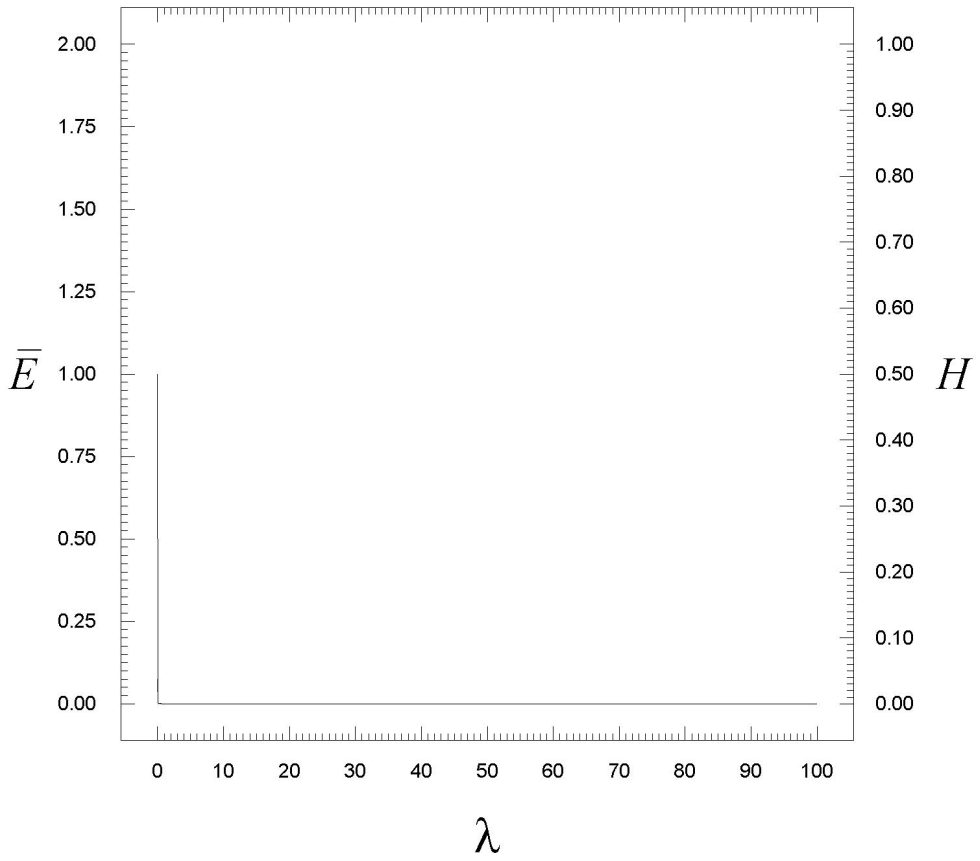


Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 14 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 1.000154 , @ lambda= 0. The maximum of H is 0.206926 , @ lambda= 35.9.

Function 15: f(A,B) = TRUE



Bifurcation diagram for 2:1:1 biased network trained on Boolean binary function 15 using the standard logistic activation function. A plot of eventual sum squared error as a function of the lambda parameter. The parameter is varied over the range [0 < lambda < 100] using a step-size of 0.1. Eventual sum squared error is plotted in range [0 < E < 4], using 100 samples for each parameter step. Weight values were initialised in the range [-0.01 < w_x < 0.01] and the network sampled after 1,000 epochs. This plot represents a total of 100,000,000 training epochs. The useable convergent gap where E <= 0.04 starts @ lambda= 0.1 and ends @ lambda= 100.



Mean error versus smoothed entropy diagram for 2:1:1 biased network trained on Boolean binary function 15 using the standard logistic activation function. A comparison of the eventual mean error E (black) with the eventual entropy of the error H (grey), as a function of lambda. Entropy of the sum squared error is calculated using 100 sample bins, smoothed, and normalised to $\log_n(100) = 1$. Mean sum squared error is plotted in the range [$0 < E < 4$]. The maximum of E is 0.998912 , @ lambda= 0. The maximum of H is 0.07525 , @ lambda= 0.

Appendix B

The Programs

Here are the program listings for the main study. All these programs are written in VisualBasic.

Firstly there are a set of programs that simulate a given network topology using the generalized delta rule. The programs are topology–specific, i.e. there are 3 simulation programs, one for a 1:1:1 network, one for a 2:1:1 network, and another for the 2:2:1 network.

Each program has the selectable parameters:

Initial condition range
Lambda parameter range
Number of epochs
Number of trials (training epochs)

Additionally the 2:1:1 program has selectable parameters to cycle training across all 16 binary Boolean functions.

Each program produces a standardized datafile for subsequent analysis. The text file format is standardised for all simulation runs.

Each processing run produces a text file of the errors recorded after E epochs, over a lambda parameter range of LMIN to LMAX in increments of LSTEP. For each lambda parameter value, there are B bins (processing runs where initial conditions are each randomized differently).
The initial conditions are psuedorandom numbers in the range IMIN to IMAX.

The filespec is aranged in three sections, with each section comprised of
CR/LF delimeted textlines as follows:

1. HEADER

TITLE	<STRING>
BINS	B
EPOCHS	E
LAMBDA-MIN	LMIN
LAMBDA-MAX	LMAX
LAMBDA-STEP	LSTEP
INIT-COND-MIN	IMIN
INIT-COND-MAX	IMAX

2. BODY

LAMBDA <FLOAT>
<FLOAT-1> <FLOAT-2> <FLOAT-3> ... <FLOAT-B>

3. EOF

Program to simulate 1:1:1 net

```
Const init_min As Double = -0.1
Const init_max As Double = 0.1
Const init_dif As Double = init_max - init_min


Function initial_cond() As Double
       initial_cond = init_min + Rnd * init_dif
End Function

' standard sigmoid activation function

Function activation_fun(x As Double) As Double
       activation_fun = 1 / (1 + Exp(-x))

End Function

' standard logistic error function

Function error_fun(x As Double) As Double
       error_fun = x * (1 - x):

End Function



Private Sub Form_Activate()


'   A program that simulates a 1:1:1 biasless network
'   The network is trained on the boolean identity function
'   The program first does a training run, and then records the
'   sum squared error of the last cycle.
'   Training passes are over a range of lambda parameter values
'   and a set number of bins for each parameter is assigned.
'   For each bin the network is initialised randomly to within a
'   specified weight range.
'   This prog produces a datafile for later processing.
'
'
'                        A       B
'        input  O-------O-------O  output
'                   wia     wab
'


Const states As Integer = 2
Const bins As Integer = 100
Const lambda_min As Double = 0
Const lambda_max As Double = 200
Const lambda_step As Double = 0.1


Dim T(states) As Double
Dim I(states) As Double
Dim n As Integer, its As Integer
Dim wia As Double, wab As Double
Dim l As Double, sse As Double
Dim A As Double, ea As Double
Dim B As Double, eb As Double
Dim trials As Integer
```

```
' setup input states and target states
' for boolean identity function

T(1) = 0:    T(2) = 1
I(1) = 0:    I(2) = 1


For trials = 100  To 500 Step 100

       file$ = "111@" + LTrim$(RTrim$(Str$(trials)))
       Form1.FontSize = 18
       Form1.Caption = "Writing " & file$
       fout$ = root$ + file$ + ".txt"

       Open fout$ For Output As #1
       Print #1, "TITLE 111 identity problem without biases"
       Print #1, "BINS "; bins
       Print #1, "TRIALS "; trials
       Print #1, "LAMBDA_MIN "; lambda_min
       Print #1, "LAMBDA_MAX "; lambda_max
       Print #1, "LAMBDA_STEP "; lambda_step
       Print #1, "INIT_CONDITION_MIN "; init_min
       Print #1, "INIT_CONDITION_MAX "; init_max

       ' generate speed-up constants
       ac = activation_fun(0)
       acc = ac * (1 - ac)

       For l = lambda_min To lambda_max Step lambda_step

              Form1.CurrentX = 0: Form1.CurrentY = 0
              Form1.Print "Lambda: ", l
              Print #1, "LAMBDA ", l

              For n = 1 To bins
                     wia = initial_cond()
                     wab = initial_cond()

                     For its = 1 To trials
                            wiai = 0
                            wabi = 0

                            For s = 1 To states
                                   A = activation_fun(I(s) * wia)
                                   B = activation_fun(A * wab)
                                   eb = error_fun(B) * (T(s) - B)
                                   ea = error_fun(A) * (eb * wab)
                                   wabi = wabi + l * eb * A
                                   wiai = wiai + l * ea * I(s)
                            Next s
                            wab = wab + wabi
                            wia = wia + wiai
                     Next its
                     sse = 0

                     For s = 1 To states
                     B = activation_fun(activation_fun(I(s)*wia)*wab)
                     sse = sse + (T(s) - B) ^ 2
                     Next s
                     Print #1, sse;
                     Next n
              Print #1,
       Next l
       Close #1
Next trials

End

End Sub
```

Program to simulate 2:1:1 net

```
Const init_min As Double = -0.01
Const init_max As Double = 0.01
Const pi As Double = 3.14159265358979
Const init_dif As Double = init_max - init_min


Function initial_cond() As Double
    initial_cond = init_min + Rnd * init_dif
End Function


' standard sigmoid activation function
Function activation_fun(x As Double) As Double
    activation_fun = 1 / (1 + Exp(-x)):
End Function


' standard logistic error function
Function error_fun(x As Double) As Double
    error_fun = x * (1 - x):
End Function



Private Sub Form_Activate()



'   211 Boolean binary n-problem with bias.
'   UNIVERSAL CONVERGENCE GAP FOUND FOR ALL BOOLEAN LOGIC FUNCTIONS
'
'    A ----
'      \   \
'       H---O---
'      /   /
'    B ----
'
'


Const funstart As Integer = 0           ' set both to 6 to do only XOR
Const funend As Integer = 15
Const states As Integer = 4
Const bins As Integer = 100
Const lambda_min As Double = 0
Const lambda_max As Double = 100
Const lambda_step As Double = 0.1



Dim s As Integer, n As Integer, i As Integer
Dim T(states) As Double, A(states) As Double, B(states) As Double
Dim wah As Double, wao As Double, wbh As Double, wbo As Double
Dim who As Double, bh As Double, bo As Double
Dim wahi As Double, waoi As Double, wbhi As Double, wboi As Double
Dim whoi As Double, bhi As Double, boi As Double
Dim l As Double, sse As Double, h As Double, o As Double
Dim eh As Double, eo As Double, aver As Double
Dim fun As Integer
Dim trials As Integer
Dim neth As Double
Dim neto As Double
```

```
' setup input states
A(1) = 0:   A(2) = 1:   A(3) = 0:   A(4) = 1
B(1) = 0:   B(2) = 0:   B(3) = 1:   B(4) = 1


For trials = 250 To 500 Step 250


       ' create a net-dynamics .txt file
       root$ = "c:\temp\"
       file$ = "211-allbools@" + LTrim$(RTrim$(Str$(trials)))
       fout$ = root$ + file$ + ".txt"
       Form1.Caption = "Writing " & fout$
       Form1.FontSize = 24


       ' generate header
       Open fout$ For Output As #1
       Print #1, "TITLE 211 biased network trained on all booleans
using standard logistic."
       Print #1, "BINS "; bins * (funend + 1 - funstart)
       Print #1, "TRIALS "; trials
       Print #1, "LAMBDA_MIN "; lambda_min
       Print #1, "LAMBDA_MAX "; lambda_max
       Print #1, "LAMBDA_STEP "; lambda_step
       Print #1, "INIT_CONDITION_MIN "; init_min
       Print #1, "INIT_CONDITION_MAX "; init_max



       For l = lambda_min To lambda_max Step lambda_step

              Print #1, "LAMBDA ", l

              ' set up boolean function targets
              For fun = funstart To funend

                     T(1) = (fun And 1)
                     T(2) = (fun And 2) / 2
                     T(3) = (fun And 4) / 4
                     T(4) = (fun And 8) / 8
                     Form1.CurrentX = 0
                     Form1.CurrentY = 0
                     Form1.Print "Function:"; fun;
                     Form1.Print "Lambda:";Int(0.5+l*1000)/1000;


                     For n = 1 To bins

                            wah = initial_cond()
                            wao = initial_cond()
                            wbh = initial_cond()
                            wbo = initial_cond()
                            who = initial_cond()
                            bh = initial_cond()
                            bo = initial_cond()


                            For i = 1 To trials

                                   wahi = 0
                                   waoi = 0
                                   wbhi = 0
                                   wboi = 0
                                   whoi = 0
                                   bhi = 0
                                   boi = 0
```

```
                                        For s = 1 To states
                                        neth=bh+wah*A(s)+wbh*B(s)
                                        h = activation_fun(neth)
                                        neto=bo+wao*A(s)+wbo*B(s)+who * h
                                        o = activation_fun(neto)
                                        eo = error_fun(o) * (T(s) - o)
                                        eh = error_fun(h) * eo * who
                                        wahi = wahi + eh * A(s)
                                        waoi = waoi + eo * A(s)
                                        wbhi = wbhi + eh * B(s)
                                        wboi = wboi + eo * B(s)
                                        whoi = whoi + eo * h
                                        bhi = bhi + eh
                                        boi = boi + eo
                                        Next s


                                        wah = wah + l * wahi
                                        wao = wao + l * waoi
                                        wbh = wbh + l * wbhi
                                        wbo = wbo + l * wboi
                                        who = who + l * whoi
                                        bh = bh + l * bhi
                                        bo = bo + l * boi
                                Next i
                                sse = 0

                                For s = 1 To states
                                        neth = bh+wah*A(s)+wbh*B(s)
                                        h = activation_fun(neth)
                                        neto = bo+wao*A(s)+wbo*B(s)+who*h
                                        o = activation_fun(neto)
                                        sse = sse + (T(s) - o) ^ 2
                                Next s
                                Print #1, sse;
                        Next n
                Next fun
                Print #1,
        Next l
        Close #1
Next trials

End

End Sub
```

Program to simulate 2:2:1 net

```
Const init_min As Double = -0.1
Const init_max As Double = 0.1
Const init_dif As Double = init_max - init_min


Function initial_cond() As Double
       initial_cond = init_min + Rnd * init_dif
End Function

' standard sigmoid activation function

Function activation_fun(x As Double) As Double
       activation_fun = 1 / (1 + Exp(-x)):
End Function

' standard logistic error function

Function error_fun(x As Double) As Double
       error_fun = x * (1 - x):
End Function



Private Sub Form_Activate()


'    221 XOR PROB w/BIAS
'
'    A ----h1
'       \ /  \
'        X    O
'       / \  /
'    B ----h2



Const states As Integer = 4
Const bins As Integer = 100
Const trials As Integer = 250
Const lambda_min As Double = 0
Const lambda_max As Double = 100
Const lambda_step As Double = 0.1


Dim s As Integer
Dim n As Integer
Dim i As Integer
Dim T(states) As Double
Dim A(states) As Double
Dim B(states) As Double
Dim wah1 As Double, wah2 As Double, wbh1 As Double, wbh2 As Double
Dim wh1o As Double, wh2o As Double, wth1 As Double, wth2 As Double,
Dim wto As Double
Dim wah1i As Double, wah2i As Double, wbh1i As Double, wbh2i As Double
Dim wh1oi As Double, wh2oi As Double, wth1i As Double
Dim wth2i As Double, wtoi As Double
Dim l As Double, sse As Double, h1 As Double, h2 As Double
Dim o As Double
Dim eh1 As Double, eh2 As Double, eo As Double
```

```
' setup target states for xor
T(1) = 0:   T(2) = 1:   T(3) = 1:   T(4) = 0
A(1) = 0:   A(2) = 1:   A(3) = 0:   A(4) = 1  ' setup input states
B(1) = 0:   B(2) = 0:   B(3) = 1:   B(4) = 1  '


' create a net-dynamics .txt file
root$ = "c:\temp\"
file$ = "221@250"
fout$ = root$ + file$ + ".txt"
Form1.Caption = "Writing " & fout$
Form1.FontSize = 24


' generate header
Open fout$ For Output As #1
Print #1, "TITLE 221 EXCLUSIVE-OR WITH BIASES"
Print #1, "BINS "; bins
Print #1, "TRIALS "; trials
Print #1, "LAMBDA_MIN "; lambda_min
Print #1, "LAMBDA_MAX "; lambda_max
Print #1, "LAMBDA_STEP "; lambda_step
Print #1, "INIT_CONDITION_MIN "; init_min
Print #1, "INIT_CONDITION_MAX "; init_max

For l = lambda_min To lambda_max Step lambda_step

       Form1.CurrentX = 0
       Form1.CurrentY = 0
       Form1.Print "Lambda: ", l
       Print #1, "LAMBDA ", l
        DoEvents

       For n = 1 To bins

              wah1 = initial_cond()
              wah2 = initial_cond()
              wbh1 = initial_cond()
              wbh2 = initial_cond()
              wh1o = initial_cond()
              wh2o = initial_cond()
              wth1 = initial_cond()
              wth2 = initial_cond()
              wto = initial_cond()

              For i = 1 To trials

                     wah1i = 0
                     wah2i = 0
                     wbh1i = 0
                     wbh2i = 0
                     wh1oi = 0
                     wh2oi = 0
                     wth1i = 0
                     wth2i = 0
                     wtoi = 0
```

```
                        For s = 1 To states

                        h1 = activation_fun(wth1+wah1*A(s)+wbh1*B(s))
                        h2 = activation_fun(wth2+wah2*A(s)+wbh2*B(s))
                        o = activation_fun(wto + wh1o * h1 + wh2o * h2)
                        eo = error_fun(o) * (T(s) - o)
                        eh1 = error_fun(h1) * eo * wh1o
                        eh2 = error_fun(h2) * eo * wh2o
                        wah1i = wah1i + eh1 * A(s)
                        wah2i = wah2i + eh2 * A(s)
                        wbh1i = wbh1i + eh1 * B(s)
                        wbh2i = wbh2i + eh2 * B(s)
                        wh1oi = wh1oi + eo * h1
                        wh2oi = wh2oi + eo * h2
                        wth1i = wth1i + eh1
                        wth2i = wth2i + eh2
                        wtoi = wtoi + eo

                        Next s

                        wah1 = wah1 + l * wah1i
                        wah2 = wah2 + l * wah2i
                        wbh1 = wbh1 + l * wbh1i
                        wbh2 = wbh2 + l * wbh2i
                        wh1o = wh1o + l * wh1oi
                        wh2o = wh2o + l * wh2oi
                        wth1 = wth1 + l * wth1i
                        wth2 = wth2 + l * wth2i
                        wto = wto + l * wtoi

                        Next i


                sse = 0

                For s = 1 To states
                        h1 = activation_fun(wth1+wah1*A(s)+wbh1*B(s))
                        h2 = activation_fun(wth2+wah2*A(s)+wbh2*B(s))
                        o = activation_fun(wto+wh1o*h1+wh2o*h2)
                        sse = sse + (T(s) - o) ^ 2
                Next s

                Print #1, sse;
        Next n
        Print #1,
Next l
Close #1

End
End Sub
```

Program to Analyse Datafiles from the simulation runs

```
Private Sub Form_Activate()


Dim bin() As Integer
Dim num() As Single
Dim savent() As Single
Dim savavn() As Single
Dim idx As Long
Dim acnt As Integer
Dim bins As Integer


'   The Analyser Program creates:
'   a .bmp of bifurcation diagram
'   a .bmp of normal entropy vs. mean average - grey lines for average
'   a .txt of footer info for both diagrams


smode = 4 * (Picture1.Width / 32400)

' set number of smooths for entropy measure
avgoes = 100

root$ = "c:\"

' number of datafiles to analyse
For fcount = 1 To 6
       Form1.Cls

       Select Case fcount
       Case 1
              ' filename
              indir$ = root$ + "221\": fil$ = "221@250"

              ' error  for bifurcation diagram
              emax1 = 4

              ' error for mean error/ entropy diagram
              emax2 = 2

              ' S.S.E. level for test of convergence gap
              sselevel = 0.04

       Case 2
              indir$ = root$ + "221\": fil$ = "221@500"
              emax1 = 4: emax2 = 2: sselevel = 0.04

       Case 3
              indir$ = root$ + "221\": fil$ = "221@1000"
              emax1 = 4: emax2 = 2: sselevel = 0.04

       Case 4
              indir$ = root$ + "111\": fil$ = "111@100"
              emax1 = 2: emax2 = 1: sselevel = 0.27

       Case 5
              indir$ = root$ + "111\": fil$ = "111@250"
              emax1 = 2: emax2 = 1: sselevel = 0.27

       Case 6
              indir$ = root$ + "111\": fil$ = "111@500"
              emax1 = 2: emax2 = 1: sselevel = 0.27

       End Select
```

```
For part = 0 To 15

        '
        '     read in the datafile
        '

       fin$ = indir$ + fil$ + ".txt"
       apart$ = "-fun" + LTrim$(RTrim$(Str$(part))) & "-5inch"
       fout1$ = indir$ + fil$ + "-bd" + apart$ + ".bmp"
       Form1.Caption = "Creating " & fout1$

       Open fin$ For Input As #1
       Input #1, head1$
       Input #1, head$: bins = Val(Mid$(head$, 5))
       Input #1, head$: trials = Val(Mid$(head$, 7))
       Input #1, head2$: lmin = Val(Mid$(head2$, 11))
       Input #1, head3$: lmax = Val(Mid$(head3$, 11))
       Input #1, head4$: lstep = Val(Mid$(head4$, 12))
       Input #1, head$: rmin = Val(Mid$(head$, 19))
       Input #1, head$: rmax = Val(Mid$(head$, 19))

       emin = 0
       emax = emax1

       ' compute maximum entropy from no. of bins
      maxent = Log(bins)

      bint = bins + 1


       '
       '     compute graph - 1st do standard bifurcation error map
       '

       meanflag = False
       x$ = "l": y$ = "E": z$ = ""
       GoSub draw_scale
       enscl = bins / ediff
       avscl = avgoes + 1

       '
       '     setup arrays for data abstaction of the plot
       '     plot data in file -0.5 to +0.5
       '     bin(1) ... bin(bins) : use lower bound of one
       '

       ReDim bin(bins) As Integer
       ReDim num(bins) As Single
       ReDim savent((lmax - lmin) / lstep) As Single
       ReDim savavn((lmax - lmin) / lstep) As Single


       '
       '     read in error data and draw a scatterplot on picture1
       '

       acnt% = 0
       ssestart = 0                            ' vars for calculation
       sseend = 0                              ' of sse level gap
       ssestartfound = False


       For l = lmin To lmax + lstep - 0.0000001 Step lstep

             If EOF(1) Then Exit For
             Input #1, A$
             lr = Val(Mid$(A$, 7))
             avn = 0
             xtemp = lr - lmin
```

```
        '  read off  first parts
        If part <> 0 Then
               For n% = 1 To bins * part
                      Input #1, junk
               Next n%
        End If

        sseflag = True
        For n% = 1 To bins
        Input #1, num(n%)
        Picture1.PSet (xfrom+xtemp*xscal,yfrom+(num(n%)*yscal))
        avn = avn + num(n%)
        If num(n%) > sselevel Then sseflag = False
        Next n%

        ' calculate gap where all SSE values are <= sselevel
        If sseflag = True Then
               If ssestartfound = False Then
               ' found start of gap
                      ssestartfound = True
                      ssestart = l
               Else
               ' already in gap, store end
                      sseend = l
               End If
        End If

        '  read last parts
        If part <> 15 Then
               For n% = 1 To bins * (15 - part)
                      Input #1, junk
               Next n%
        End If

        '
        '       Calculate smoothed entropy and store
        '

        ent = 0
        For goes% = 1 To avgoes

               Erase bin: ReDim bin(bins)
               ' smoothing goes from -0.5 to +0.5
               addb = -0.5 + goes% / avgoes

               For n% = 1 To bins
                      idx = 1 + Int(addb + enscl * num(n%))
                      ' wrap to option base 1
                      If idx = 0 Then idx = bins
                      If idx = bint Then idx = 1
                      bin(idx) = bin(idx) + 1
               Next n%

               For n% = 1 To bins
                      p = bin(n%) / bins
                      If p > 0 Then ent = ent - p * Log(p)
               Next n%

        Next goes%

        av = avn / bins
        en = ent / (avscl * maxent)
        savavn(acnt%) = av
        savent(acnt%) = en
        acnt% = acnt% + 1
        DoEvents
Next l
DoEvents
SavePicture Picture1.Image, fout1$
Close #1
```

```
'
'      Get 1st and last minima
'

minav = 99: minen = 99
maxav = 0: maxen = 0
acnt% = 0

For l = lmin To lmax - lstep Step lstep

       If savavn(acnt%) < minav Then
              minav = savavn(acnt%)
              mal1 = l
              sacta = acnt%
       End If
       If savent(acnt%) < minen Then
              minen = savent(acnt%)
              mel1 = l
              sacte = acnt%
       End If

       ' also get maximums
       If savavn(acnt%) > maxav Then
              maxav = savavn(acnt%)
              maxl = l
       End If
       If savent(acnt%) > maxen Then
              maxen = savent(acnt%)
              mexl = l
       End If

       acnt% = acnt% + 1
Next l


'
'      get gapwidth of lowest mean error
'
acnt% = sacta

For l = mal1 To lmax - lstep Step lstep
       If savavn(acnt%) <= minav Then
              minav = savavn(acnt%)
              mal2 = l
       Else
              Exit For
       End If
       acnt% = acnt% + 1
Next l
gapa = mal2 - mal1


'
'      get gapwidth of lowest entropy
'
acnt% = sacte
For l = mel1 To lmax - lstep Step lstep
       If savent(acnt%) <= minen Then
              minen = savent(acnt%)
              mel2 = l
       Else
              Exit For
       End If
       acnt% = acnt% + 1
Next l
gape = mel2 - mel1
```

```
        '
        '      write footnotes .txt file for both diagrams
        '

        fout2$ = indir$ + fil$ + "-info" + apart$ + ".txt"
        Form1.Caption = "Writing " & fout2$

        Open fout2$ For Output As #2
        Print #2, Mid$(head1$, 7)
        Print #2, "A plot of eventual sum squared error as a function
of the lambda parameter."
        Print #2, "The parameter is varied over the range [";
Int(lmin); "< L <"; Int(lmax); "] using a step-size of"; lstep; "."
        Print #2, "Eventual sum squared error is plotted in range [";
emin; "< E <"; emax; "], using"; bins; "samples for each parameter
step."
        Print #2, "Weight values were initialised in the range [ ";
rmin; "< Wx <"; rmax; "] and the network sampled after"; trials;
"epochs."
        Print #2, "This plot represents a total of"; Int(trials * bins
* (lmax - lmin) / lstep); "training epochs."
        Print #2, "The useable convergent gap where E <="; sselevel; "
starts @ l="; ssestart; "and ends @ l="; sseend
        Print #2,


        '
        '      footnote for entropy/mean error map
        '


        Print #2, Mid$(head1$, 7); "."
        Print #2, "A comparison of the eventual mean error E (black)
with the eventual entropy of the error H (grey), as a function of L."
        Print #2, "Entropy of the sum squared error is calculated
using"; bins; "sample bins, smoothed, and normalised to log(bins) = ";
RTrim$(Str$(Int(maxent * 1000000) / 1000000)); "."
        Print #2, "Mean sum squared error is plotted in the range [";
emin; "< E <"; emax; "]. "
        Print #2, "The maximum of E is"; Int(maxav * 1000000) /
1000000; ", @ lambda="; Int(maxl * 1000) / 1000; "."
        Print #2, "The maximum of H is"; Int(maxen * 1000000) /
1000000; ", @ lambda="; Int(mexl * 1000) / 1000; "."
        Print #2, "The minimum of E is"; Int(minav * 1000000) /
1000000; ", @ lambda="; Int(mal1 * 1000) / 1000; "to"; Int(mal2 *
1000) / 1000; ". Gap width="; gapa
        Print #2, "The minimum of H is"; Int(minen * 1000000) /
1000000; ", @ lambda="; Int(mel1 * 1000) / 1000; "to"; Int(mel2 *
1000) / 1000; ". Gap width="; gape
        Print #2, "Input File:"; fin$
        Close #2


        '
        '      plot mean error & normalised H and initialise pic & font
scales ( 2nd .BMP )
        '      if this requires different scale to E than the
bifurcation diagram, use var emax2
        '

        fout3$ = indir$ + fil$ + "-em" + apart$ + ".bmp"
        Form1.Caption = "Creating " & fout3$
        ' default symbols for axis labels
        x$ = "l"
        y$ = "E"
        z$ = "H"
        meanflag = True
        emax = emax2
        GoSub draw_scale
```

```
        ' plot entropy in grey

        Picture1.ForeColor = RGB(128, 128, 128)
        acnt% = 0

        For lr = lmin To lmax + 0.00001 Step lstep
                ex = xfrom + xscal * lr
                ey = yfrom + yscal * savent(acnt%)
                acnt% = acnt% + 1
                If acnt% <> 1 Then
                        Picture1.Line -(ex, ey)
                Else
                        Picture1.CurrentX = ex
                        Picture1.CurrentY = ey
                End If
        Next lr

        ' plot mean in black

        Picture1.ForeColor = RGB(0, 0, 0)
         acnt% = 0

        For lr = lmin To lmax + 0.00001 Step lstep
                ex = xfrom + xscal * lr
                ey = yfrom + yscal * savavn(acnt%)
                acnt% = acnt% + 1
                If acnt% <> 1 Then
                        Picture1.Line -(ex, ey)
                Else
                        Picture1.CurrentX = ex
                        Picture1.CurrentY = ey
                End If
        Next lr
        DoEvents
        SavePicture Picture1.Image, fout3$
        Next part
Next fcount

End
End Sub


'
'       graphical subroutines
'

draw_scale:

        ediff = emax - emin
        If emax = 1 Then ydivs = 100 Else ydivs = 80
        yfrom = 1 / 20
        yto = 1 - yfrom
        ydist = yto - yfrom
        If lmax = 4 Then xdivs = 80 Else xdivs = 100
        xfrom = 1 / 20
        xto = 1 - xfrom       ' adjusts how far in first division goes
        xdist = xto - xfrom
        xst = xdist / xdivs
        do_graph x$,y$,z$,lmin,lmax,emax,xdivs,ydivs,smode,meanflag
        yscal = ydist / ediff
        xscal = xdist / (lmax - lmin)
Return
```

```
Sub do_graph(x$, y$, z$, lmin, lmax, emax, xdivs, ydivs, smode,
meanflag)


        Dim CX, CY, Limit, Radius   As Integer, Msg As String
        Picture1.Scale (-0.2, 1.1)-(1.2, -0.3)
        Picture1.Cls
        Picture1.DrawStyle = vbSolid
        Picture1.FillStyle = vbFSSolid
        Picture1.FillColor = QBColor(0)
        AutoRedraw = True
        Picture1.FontSize = Int(8 * smode)
        Picture1.FontName = "Arial"
        Picture1.FontSize = Int(8 * smode)
        emin = 0


        If lmax = 4 Then xsubdc = 8 Else xsubdc = 10
        xdivmod = xdivs / xsubdc   ' subdivisions
        xfrom = 1 / 20             ' start 1/20 total in from borders
        xto = 1 - xfrom
        xdist = xto - xfrom
        xst = xdist / xdivs

        If emax = 1 Then ysubdc = 10 Else ysubdc = 8
        ydivmod = ydivs / ysubdc
        yfrom = 1 / 20
        yto = 1 - yfrom
        ydist = yto - yfrom
        yst = ydist / ydivs


        ' draw borders
        Picture1.Line (0, 0)-(0, 1)
        Picture1.Line (0, 1)-(1, 1)
        Picture1.Line (1, 1)-(1, 0)
        Picture1.Line (1, 0)-(0, 0)


        ' x axis graduated scale (L)
        docnt = 0
        For X1 = xfrom To xto + xst / 2 Step xst

              Picture1.Line (X1, 1)-(X1, 1 - 0.01)          ' top
              Picture1.Line (X1, 0)-(X1, 0.01)              ' bot
              ' txt label every other one with lvalue [lmax,lmin]
              docnt = docnt + 1
              If (docnt Mod xdivmod) = 1 Then
                    Picture1.Line (X1, 1)-(X1, 1 - 0.02)    'top
                    Picture1.Line (X1, 0)-(X1, 0.02)        'bot
                    l = lmin + ((lmax - lmin) * (X1 - xfrom) / xdist)
                    Msg = LTrim$(RTrim$(Str$(Int(0.05+100*l)/100)))
                    If 0 = (InStr(Msg, ".")) Then Msg = Msg & ".0"
                    Picture1.CurrentX = Picture1.CurrentX -
                    Picture1.TextWidth(Msg) / 2
                    Picture1.CurrentY = Picture1.CurrentY +
                    Picture1.TextHeight(Msg) * 1.5
                    Picture1.Print Msg
              End If
        Next X1
```

```
	' do left graduated scale (E)

	docnt = 0
	For Y1 = yfrom To yto + yst / 2 Step yst

	If z$ <> "H" Then Picture1.Line (1, Y1)-(1 - 0.01, Y1)
		Picture1.Line (0, Y1)-(0.01, Y1)
		docnt = docnt + 1
		If (docnt Mod ydivmod) = 1 Then
			' txt evalue [emax,emin]
			If z$<>"H" Then Picture1.Line (1,Y1)-(1-0.02,Y1)
			Picture1.Line (0, Y1)-(0.02, Y1)
			E = emin + ((emax - emin) * (Y1 - yfrom) / ydist)
			Msg = LTrim$(RTrim$(Str$(Int(0.0005+100*E)/100)))
			If 0 = (InStr(Msg, ".")) Then Msg = Msg & ".0"
			If (1 = InStr(Msg, ".")) And (Len(Msg) = 2) Then
Msg = Msg & "0"
			If (2 = InStr(Msg, ".")) And (Len(Msg) = 3) Then
Msg = Msg & "0"
			If Mid$(Msg, 1, 1) = "." Then Msg = "0" & Msg
			Picture1.CurrentX = Picture1.CurrentX -
Picture1.TextWidth("0000000")
			Picture1.CurrentY = Picture1.CurrentY -
Picture1.TextHeight(Msg) / 2
			Picture1.Print Msg
		End If
	Next Y1


	' if z$="H" then do right graduated scale (H)

	If z$ = "H" Then
		ydivs = 100
		ydivmod = ydivs / 10
		yst = ydist / ydivs	' subdivisions always 10
		docnt = 0

		For Y1 = yfrom To yto + yst / 2 Step yst

			Picture1.Line (1, Y1)-(1 - 0.01, Y1)
			docnt = docnt + 1
			If (docnt Mod ydivmod) = 1 Then

				Picture1.Line (1, Y1)-(1 - 0.02, Y1)
				h = (Y1 - yfrom) / ydist
				Msg=LTrim$(Str$(Int(0.0005+100*h)/ 100))
				If 0=InStr(Msg,".") Then Msg = Msg & ".0"
				If 1=InStr(Msg,".") And Len(Msg)=2 Then Msg
= Msg & "0"
				If 2=InStr(Msg,".") And Len(Msg)=3 Then Msg
= Msg & "0"
				If Mid$(Msg,1,1) = "." Then Msg="0" & Msg
				Picture1.CurrentX = 1 +
Picture1.TextWidth("00")
				Picture1.CurrentY = Picture1.CurrentY -
Picture1.TextHeight(Msg) / 2
				Picture1.Print Msg
			End If
		Next Y1
	End If
```

```
' label axes with symbols

       Picture1.FontSize = Int(18 * smode)
       Picture1.FontName = "Times New Roman"
       Picture1.FontItalic = True
       Picture1.FontSize = Int(18 * smode)

       Msg = y$
       Picture1.CurrentY = 0.5 - Picture1.TextHeight(Msg) / 2
       Picture1.CurrentX = -Picture1.TextWidth(Msg) * 4
       Picture1.Print Msg;

       If meanflag = True Then
              Msg = "_"
              Picture1.CurrentY = 0.5 + 0.065 -
Picture1.TextHeight(Msg) / 2
              Picture1.CurrentX = -Picture1.TextWidth(Msg) * 4.5
              Picture1.Print Msg;
       End If

       Msg = z$
       Picture1.CurrentY = 0.5 - Picture1.TextHeight(Msg) / 2
       Picture1.CurrentX = 1 + Picture1.TextWidth(Msg) * 2.5
       Picture1.Print Msg;
       Picture1.FontSize = Int(20 * smode)
       Picture1.FontName = "Symbol"
       Picture1.FontItalic = False
       Picture1.FontSize = Int(20 * smode)

       Msg = x$
       Picture1.CurrentY = Picture1.TextHeight(Msg) * 1
       Picture1.CurrentX = 0.5 - Picture1.TextWidth(Msg) / 2
       Picture1.Print Msg
       Picture1.FontName = "Times New Roman"

End Sub
```