



Global Optimisation and Network Training

John McKay

john@oscarkilo.net

Abstract

The global optimisation of an energy function and the attendant problem of local minima have been the subject of extensive research. Recently, several methods have emerged which can guarantee to find the global minimum within a bounded region. We concentrate on one such method, due to Chao [1], and show how it may be adapted to overcome the susceptibility of conjugate gradient descent to get stuck in local minima. The ability to deal with local minima then prompts a brief discussion of a generalised neuron model which is not, for example, subject to the 'semi-linear' restriction of Rumelhart and McClelland.

Keywords

Artificial Neural Networks, Backpropagation, Conjugate Gradient, Homotopy, Global Minimum, Neuron Models.

1. Introduction

Cybenko[2] has shown that a multilayer Artificial Neural Network (ANN), when it has just two sufficiently large hidden layers, can approximate any non-linear function as accurately as desired, and that it can approximate any continuous non-linear function with just one hidden layer. This versatility has led to the use of multilayer ANNs in applications as varied as non-linear regression analysis, monitoring and control, and classification problems. Whatever the application, the training method is generally a variant of gradient descent being used to minimise an energy function which expresses the accuracy of the network's output, and with the local gradient being computed by backpropagating the error through the network. Unfortunately, this method can be computationally very expensive and there is no guarantee that it will converge to the optimal solution as it may not find the global, or even near optimal minimum, but get stuck in a sub-optimal local minimum. Perversely, methods such as conjugate gradient descent, which are computationally much more efficient tend to be more prone to entrapment by local minima.

Over the past decade there has been considerable research into methods of finding the global minimum of a potential energy field. The problem is mathematically ill-defined leading to the assumption that the only way to guarantee finding the global minimum is to perform an exhaustive search of all the local minima. Many methods have been proposed. Simulated Annealing is probably the most widely known, but although global convergence can be proven, such methods can be impractically slow. Genetic Algorithms, particularly the so-called structured GAs, have been used to perform a global search of the weight space, but once again this method can be very slow and is better

suited to exploring 'difficult' parameter spaces such as network topology. The methods of statistical modelling can be used, where the weights of the network are regarded as the parameters of the model. One such method is that suggested by Jabobs[3]. It is argued that these statistical methods scale better with problem size as the problem of local minima in large multi-layer perceptrons can lead to very long training times.

Recently, Chao[1] has described a method of enumerating all the extrema (maxima and minima) in a bounded region. The main thrust of this paper is to describe the essentials of this method (section 4) and to show how it may be adapted for use with a conjugate gradient decent scheme (section 5). However, before discussing the details of the method it is necessary to justify the enumeration of local minima and the pursuit of the global minimum. To this end the problem of network training is cast as an optimisation problem in section 3.

Having established that local minima need not present an insurmountable obstacle to learning by gradient descent methods, a generalisation of the normal neuron model is introduced (section 6). These more general models, which can cause the error surface to form large local minima that are far from optimal, would be totally unsuitable for a simple gradient descent approach but possess certain advantages when the local minima problem can be ignored. Dawson and Schopfloch [4] have shown that by modifying the energy function it is possible to train networks with non-monotonic activation functions and show that often these networks are easier to train than the conventional 'semi-linear' networks. The use of polynomial activation functions and the subsequent reduction in network complexity is discussed by Piazza[11].

2. Gradient Descent

Let \mathbf{i} be the input vector to the network, \mathbf{w} be the network weights and \mathbf{o} be the output vector of the network, then a feedforward network may be simply expressed as: $\mathbf{o} = \mathbf{N}(\mathbf{w}, \mathbf{i})$

In order to train the network $\mathbf{N}(\mathbf{w}, \mathbf{i})$ we must first define a function which in some way measures how well the network is trained. This function is usually constructed so that it measures the error between the required output of the network and the actual output. For example, the following functions both serve this purpose:

$$E_p = \sum_{i=1}^L (t_i - o_i)^2 \quad \text{The quadratic energy function}$$

$$E_p = \sum_{i=1}^L \left[\frac{1}{2}(1+t_i) \ln \frac{1+t_i}{1+o_i} + \frac{1}{2}(1-t_i) \ln \frac{1-t_i}{1-o_i} \right] \quad \text{The entropic energy function (see [5b,$$

5c])

and $E = \sum E_p$ where E_p is the network energy for a particular training pattern (vector).

Further terms may be added to the energy function which might, for instance, penalise excessive growth of the weights or equally will optimise the Minimum Descriptive Length of the network weights. (see section 3)

The energy function can be viewed as describing a surface in 'L'-dimensional space. To train the network it is necessary to adjust the network weights so that the energy function is minimised by moving from our start position in weight space, downhill until we reach our desired goal. This can be accomplished in several ways; incrementally, by steepest descent or by conjugate gradient descent. Unfortunately, there may be many local minima which may stop the process reaching an acceptable solution. The most widely used approach is the incremental, on-line method (see next section) defined below:

$$\Delta \mathbf{w} = - \eta \text{grad}(E) \quad \text{where} \quad \text{grad}(E) = \sum_{ij} \frac{\partial E}{\partial w_{ij}} \hat{i}_{ij} \quad i = \text{inputs}, j = \text{nodes} \quad (2.1)$$

and η = is a proportionality constant called the learning rate

2.1 Batch Mode Learning

Often it is desirable to present an input-output vector pair for learning, calculate the network energy, determine the gradient for this data and adjust the weights accordingly. This is then repeated for each pair in the training set. This is on-line learning and is the technique most widely used. The advantage of this method is that the energy surface will be different for each training pair and so the probability of entrapment by local minima is reduced. Also, by ensuring that the order in which the training pairs are drawn from the training set during an epoch is random, limit cycles in the weight updates can be avoided. This approach can be very slow but the rather wandering nature of the search does explore the weight space and can yield a network with good generalisation characteristics.

In contrast, during batch mode learning the weights are updated after all the training pairs have been presented. This results in an energy surface which is consistent from one training epoch to the next. This is a double-edged sword, as the consistency of the energy surface from one epoch to the next increases the probability of getting trapped in a local minimum, while on the other hand it is this very consistency that allows the use of a true conjugate gradient decent. It is the efficiency of conjugate gradient methods combined with a more systematic exploration of the weight space that motivates the method that is described in sections 4 and 5. *In all that follows it is assumed that batch mode learning is being used.*

2.2 Conjugate Gradient Descent

This is an elegant variant of the steepest descent method. Briefly, the steepest descent approach seeks to find the minimum along a line which is in the direction of steepest decent. At the line minimum this is repeated and so the search continues in a direction that will be orthogonal to the last direction. This is not necessarily the best course as it constrains the search to a zig-zag route. Conjugate gradient descent corrects this flaw by calculating a new search direction which is conjugate to it at that point. By minimising along the line which is conjugate the least disruption to the minimisation achieved along the previous search lines is ensured. As in the case of steepest descent the minimisation

is now a series of one dimensional line minimisations and so the armoury of methods that are capable of fast convergence can be used.

The conjugate direction $\hat{\mathbf{e}}_{k+1}$ is calculated using the Polak-Ribiere equation. A very lucid account of this method is presented in [5a].

$$\hat{\mathbf{e}}_{k+1} = -\nabla E(\mathbf{w}_{k+1}) + \beta \hat{\mathbf{e}}_k \quad \text{and} \quad \beta = \frac{(\nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k)) \cdot \nabla E(\mathbf{w}_{k+1})}{\nabla E(\mathbf{w}_k) \cdot \nabla E(\mathbf{w}_k)} \quad (2.2)$$

2.1.1 The Line Minimisation

There are many methods for finding the minimum of a 1-D function. The method described here was chosen as it is particularly suitable when considering the elimination of extrema as described in section 5.2.

To minimise along the line we first take a small step, s_δ , downhill (in the direction of the negative gradient) and the gradient is then re-evaluated. The next step is calculated using the formula below. If the energy function along the line $E(s)$ is parabolic in form, then this step will take us directly to the minimum. In practice $E(s)$ is unlikely to be so co-operative and successive approximations will be necessary. To ensure that unreasonably large steps are not taken, for instance when moving through regions where the local curvature is small relative to the local gradient, a limit, s_+ , is placed on the size of each step.

$$s_0 = \text{sgn}(g_0) s_\delta \quad \text{where} \quad s_\delta = \text{a small constant } O(10^{-4})$$

$$s_k = -\frac{dE}{ds} \bigg/ \left| \frac{d^2E}{ds^2} \right| = g_k / c_k \quad \text{where} \quad c_k = |(g_k - g_{k-1}) / s_{k-1}| \quad \text{and} \quad |s_k| \leq s_+ \quad (2.3)$$

The constants are typically $s_\delta = 0.0001$ and $s_+ = 0.5$

2.3 Network Topology and the Energy Surface Topology

The topology of the energy surface depends on the network's topology and of course the energy metric in use. Below is a brief summary of the characteristics of the energy surface generated by the quadratic energy function for various ANN topologies: (see section 5.1)

- A single layer ANN with linear transfer functions has only one minimum. The profile of the minimum will be that of a parabola.
- A single layer ANN with non-linear monotonic transfer functions has only one minimum. The shape of the minimum will no longer be parabolic
- A multilayer ANN with non-linear monotonic transfer functions may have many minima. (note: a multilayer network with linear transfer functions can always be reduced to a single layer)

From the above it can be seen that gradient descent is guaranteed to find the optimum solution in weight space for the first two cases but may not find the optimum solution in the case of the multilayer ANN as it could become trapped in a non-optimal minima.

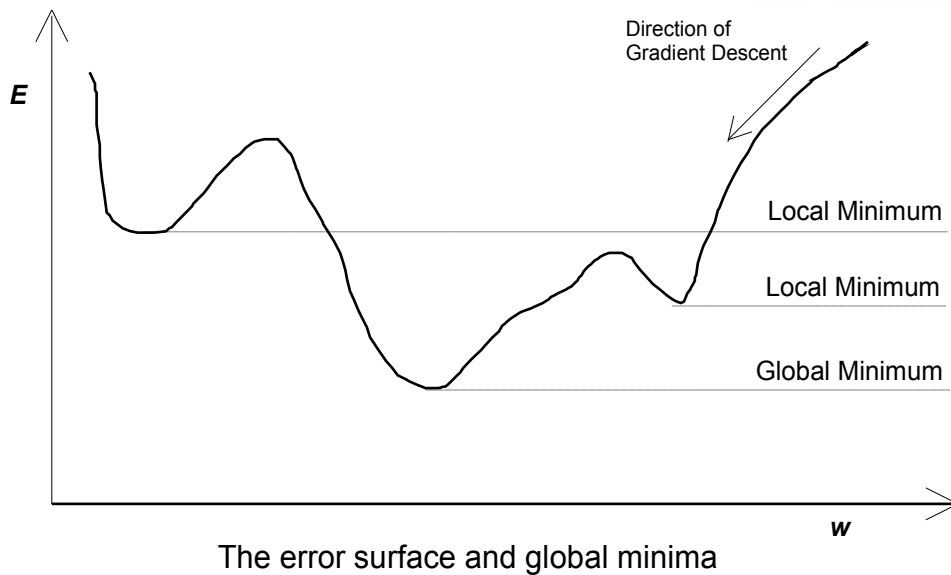


Fig 1

2.4 Finding the Gradient Field by Backpropagation

We include here a very brief discussion of backpropagation as it is central to much that follows. However the reader is referred to [6] for a complete explanation.

Backpropagation is a method of calculating the gradient of the energy surface for a particular point \mathbf{w} in weight space. Having found this gradient vector, the principles of gradient descent may be applied to reduce the network's energy.

Let $o_j = A(\text{net}_j) = A\left(\sum_i w_{ij}i_i\right)$ be the model for neuron j and let $E = E(\mathbf{w})$ (2.4)

By the chain rule: $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}}$ then as $\frac{\partial \text{net}_j}{\partial w_{ij}} = o_i$ and defining $\delta_j = -\frac{\partial E}{\partial \text{net}_j}$

we can write

$$\boxed{\frac{\partial E}{\partial w_{ij}} = -\delta_j o_i} \quad (2.5)$$

Now $\delta_j = -\frac{\partial E}{\partial \text{net}_j} = -\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = -\frac{\partial E}{\partial o_j} A'(\text{net}_j)$ since $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial A(\text{net}_j)}{\partial \text{net}_j} = A'(\text{net}_j)$ (2.6)

It can be seen from 2.5 that the problem of finding the gradient is reduced to finding a value for δ at each node j . Equation 2.6 shows that to evaluate these δ it is necessary to evaluate the term $\partial E / \partial o$. To do this we observe that $\partial E / \partial o$ can be written:

$$\frac{\partial E}{\partial o_j} = \sum_k \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial o_j} \quad (2.7)$$

This expresses the fact that the change in the network energy E due to a change in the output of node j is equal to the sum of the changes in the network energy caused by changes in the value of net in all those nodes to which the output j is connected. If the

output j is not connected to a particular node, its contribution to this sum will be zero as $\partial net/\partial o$ will be zero. In a feedforward network only those nodes that are 'after' the node j will contribute.

Using our definition of delta and noticing that $\frac{\partial net_k}{\partial o_j} = w_{kj}$ we can rewrite equation (2.7) as

$$\frac{\partial E}{\partial o_j} = \sum_k \delta_k w_{jk} \text{ and hence equation (2.6) becomes: } \boxed{\delta_j = -A'(net_j) \sum_k \delta_k w_{jk}} \quad (2.8)$$

As we have seen, once all the deltas are known for the nodes in the layer to which the output of the node j is connected then the delta for node j can be evaluated using 2.8 and hence the gradient of E with respect to its input weights can be calculated using 2.5. So once the process is started we can pass backwards through the network evaluating the gradient. To start this process off we observe that:

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -(T_j - O_j)A'(net_j) \text{ for the output layer if } E = \frac{1}{2}(T - O)^2$$

3 Optimisation and Network Training

The training of ANNs is usefully expressed as a constrained optimisation problem. The *problem* is to train the network to minimise the error in its output for a given set of inputs and the *constraint* is to minimise the error in its output for another 'related but distinct', and unseen, set of inputs. To solve this constrained optimisation problem we seek to incorporate the constraint within the problem and to solve the resulting unconstrained problem.

The *problem* is most often formalised by seeking to minimise a cost or energy function which expresses the error in the network's output. A common choice is the Root Mean Square error (see section 2) as it is a 'maximum likelihood estimator' for the network weights (cf. model parameters), assuming the data is subject to a gaussian noise.

The *constraint* is not so easy to formalise. The ability of the trained network to generalise to unseen data depends both on the data used to train it and the network's complexity. The training set must be a sufficiently large sample to characterise adequately the process generating the data while the network's complexity must be such that there are sufficient degrees of freedom (weights) to model this process. Should there be too few free parameters, the network will be unable to absorb the training set and the network will underfit the data, while too many will mean the network will tend to overfit the data. Where there is too little information in the training data to fully specify the parameters of a model, the problem is said to be ill-posed. To transform an ill-posed problem into a well-posed one, and thus to avoid overfitting, the solution needs to be stabilised by imposing some further constraints. How this may be done is the subject of Tikhonov's regularisation theory. The theory introduces the idea of a model complexity penalty function which is added to the cost function. It is then hoped that this extra term will render the problem well-posed. The use of penalty functions is a standard method of converting a constrained optimisation problem into an unconstrained problem to facilitate its solution and this is exactly what the addition of the complexity penalty term does. The form of this penalty function depends on the problem domain, but in general, in the context of ANNs, those network connections which are irrelevant to the successful

training of the network are disabled by their associated weights being driven to zero. This reduces the networks complexity and hence its tendency to overfit the data and thereby increases its ability to generalise to unseen data. Obviously, for this strategy to work we need to start with a network that has too many weights. Having decided on the size of the training set it is common to set the number of weights equal to the training set size. The number of hidden layers and how these weights should initially be distributed between them can only be deduced from a knowledge of the process being modelled. Several formal methods of including a priori information in the design of the network have been proposed [7].

3.1 Model Complexity Penalty Terms

There are innumerable approaches to the construction of penalty terms. Many are essentially equivalent, drawing their inspiration from information theory or work by Rissanen on Minimum Descriptive Length [8]. A useful review of some of these can be found in [9]. A method not covered in [9] is a method due to Bishop [10]. We mention here just a couple of very simple penalty terms for completeness.

Let $E' = E + \mu P$ where E is the unmodified energy and P is the penalty term. μ is known as 'the regularisation parameter' and determines the relative importance of the penalty term.

One of the simplest penalty terms is $P = \sum_i w_i^2$ which simply requires the sum of all the weights to be minimised. This tends to penalise large weights. A better penalty term is

$$P = \sum_i \frac{(w_i/w_0)^2}{1+(w_i/w_0)^2}$$

3.2 Training strategy

Having generated a suitable training set and designed the initial network the next step is to minimise the energy function.

If the energy function has no penalty term, then while it is being minimised the network is validated against another set of test data after each learning epoch. Initially the ability of the network to generalise to the test data will improve. However, before the energy function has reached a minimum, when evaluated against the training data, it will begin to increase against the test data. At this point overfitting or underfitting of the training data is beginning and training should be stopped. The network, as formulated by its designer, is now optimally trained. However, there are two points to note. Firstly, the network's complexity is not optimal for the problem being solved otherwise overfitting/underfitting would not be expected. Secondly, the optimal solution for this network is not coincident with, or necessarily even close to, a minimum of the cost function.

If on the other hand the energy function does include a model complexity penalty term which has been properly designed then, in contrast to the above, one of the minima of the energy function will yield an optimal network. However, there may be several such minima and the relative importance of each will depend on the relative importance given to the penalty term in the cost function. It is this that makes the method described in the next section a useful tool for training networks in real-world applications and justifies the enumeration of all the minima.

4. Removing Local Minima by Deforming the Gradient Field

We now describe this sophisticated technique described by Chao[1]. This method guarantees to find the global minimum within a bounded region.

4.1 A 1-D Illustration.

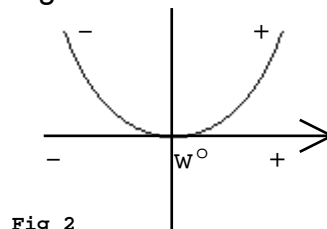
Consider the energy function E below which has extrema at $w = +2, +1, 0, -1, -2$. Starting from $w = 3$, the direction of the negative gradient is followed until the first minimum is reached at $w = 2$. Then we deform the gradient field to remove this extremum from the energy field and proceed down the gradient of the deformed field until we reach the next extremum, a maximum, at $+1$. This process may be repeated until all the extrema have been eliminated. This procedure is illustrated below:

<u>Energy</u>	<u>Gradient</u>	<u>Extrema Removed</u>
$E_0 = 2w^6 - 15w^4 + 24w^2$	$\frac{dE_0}{dw} = 12w(w^2 - 1)(w^2 - 4)$	
$E_1 = \frac{12}{5}w^5 + 6w^4 - 4w^3 - 12w^2$	$\frac{dE_1}{dw} = \frac{dE_0}{dw} / (w - 2) = 12w(w^2 - 1)(w + 2)$	min at $w = 2$
$E_2 = 3w^4 + 12w^3 + 12w^2$	$\frac{dE_2}{dw} = \frac{dE_1}{dw} / (w - 1) = 12w(w + 1)(w + 2)$	max at $w = 1$
$E_3 = 4w^3 + 18w^2 + 24w$	$\frac{dE_3}{dw} = \frac{dE_2}{dw} / w = 12(w + 1)(w + 2)$	min at $w = 0$
$E_4 = 6w^2 + 24w$	$\frac{dE_4}{dw} = \frac{dE_3}{dw} / (w + 1) = 12(w + 2)$	max at $w = -1$
$E_5 = 12w$	$\frac{dE_5}{dw} = \frac{dE_4}{dw} / (w + 2) = 12$	min at $w = -2$

Notice that the procedure alternates between finding maxima and minima of the energy function E .

To eliminate an extremum from the gradient field we have deformed it according to:

$\frac{dE_{k+1}}{dw} = \frac{dE_k}{dw} / (w - w_k^0)$. By figure 2 it can be seen that with this deformation, when local to w_k^0 , the gradient will be positive (assuming the second derivative is not zero). Thus we will always continue searching in the direction of decreasing w with such a deformation.



4.2 Mapping to an N-D Torus

If the search started at $x = 1/4$ it will find the turning points at 0, -1 and -2 but not those at +2 and +1. To ensure that it does we map the infinite real line x to the circle by the formula:

$x^{new} = (x - L) \bmod (U - L) + L$ where L is a lower bound on x and U is an upper bound. As the search proceeds it will reach the lower bound and then continue to search from the upper bound again and so find all the extrema.

This idea can be generalised to N-dimensions. Then the mapping defined by $x_j^{new} = (x_j - L_j) \bmod (U_j - L_j) + L_j$ is between the hypercube $L_j \leq x_j \leq U_j$ and the N-Torus, as illustrated in Fig 3 for the two dimensional case.

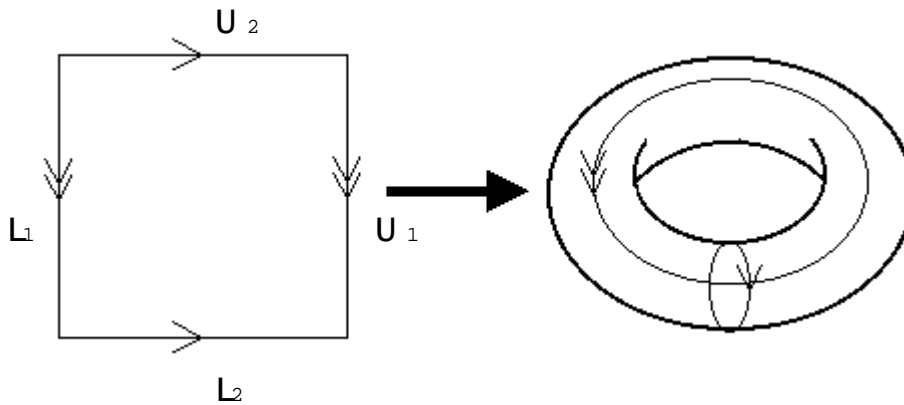


Fig 3

4.2.1 Crossing the boundary

When crossing the boundary, consideration has to be given to the gradient vector.

Assume that the k^{th} gradient field is given by $\frac{dE_k}{dw} = 6w(1-w)$ {ie. $E_k = 3w^2 - 2w^3$ }

If we now eliminate the extremum at $w = 0$, and take a step downhill from $w = 0$ in the $k+1^{th}$ gradient field $\frac{dE_{k+1}}{dw} = \frac{dE_k}{dw} / w = 6(1-w)$ {ie. $E_{k+1} = 6w - 3w^2$ } then from Fig 4 it can be seen that we shall move towards the boundary. As the boundary is crossed, the gradient will change from $6(1-L)$ to $6(1-U)$. Now if $L < 1$ and $U > 1$ then there will be a change of sign to the gradient field as it crosses the boundary, which will cause the search to change direction and head back to the boundary and hence never find the last remaining turning point. This cannot be allowed to happen, so we must ensure that the gradient does not change sign as the boundary is crossed. Let the gradient field be described by the function g . We can then ensure continuity of the sign of g , as the boundary is crossed, by multiplying g by s where $s = \text{sgn}[g(L)]\text{sgn}[g(U)]$

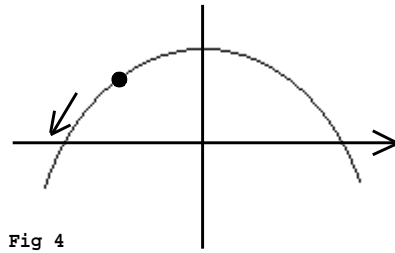


Fig 4

In general therefore, s is initially set to 1 and the gradient field is always multiplied by s . Every time the boundary is crossed s is recalculated by $s^{new} = s^{old} \operatorname{sgn}[s^{old} g(L)] \operatorname{sgn}[s^{old} g(U)]$ and hence $g_k = s^{new} g_k$.

5. Higher Order Neuron Models

As was noted in the introduction, ANNs composed of the generalised neurons discussed below can lead to highly convoluted energy surfaces. However, the method described in section 4 effectively releases us from the tyranny of local minima. So, although the generalised neuron model may induce many such minima in the energy surface this need not be a deciding factor in their use.

The neuron model considered so far has been the standard model where inputs are passed as a linear weighted sum to a static activation function to form an output. Here we consider the effect of generalising the standard model to allow both the weighting of inputs to become polynomial and the activation function to become adaptive.

$$\text{More formally, we let } o_j = A\left(\sum_k p_k(i_k), \alpha_{1j}, \dots, \alpha_{mj}\right) \quad (5.1)$$

where A is the activation function, α is the adaptive parameter(s) and p is an input polynomial function such that $net_j = \sum_i p_{ji}$ where $p_{ji} = \sum_k w_{jik} o_i^k \quad k \in [1, n]$ (5.2)

This generalisation leads to many more free parameters per neuron for a network with the same connectivity and a very different relationship between these parameters and the network energy. The greater free parameters may lead to a smaller networks but the prime motivation behind this approach is that the topology of the energy surface may be more amenable to gradient descent for particular applications.

In many circumstances it is possible to reduce a network's complexity [11] by introducing trainable parameters into the activation function. A trivial example is to allow the activation function to take the form: $o_j = \tanh(c_{j1}w + c_{j2}w^2 + c_{j3}w^3)$ then only a single node, without input weights, is required to solve the XOR problem. The simplicity of the resulting network often leads to very fast training times. In this example only 4 iterations were required to reduce the error to $O(10^{-4})$.

The resulting activation function may not be monotonic. This will not necessarily lead to many local minima in the parameter sub-space to which the polynomial coefficients belong but may lead to an increase in the complexity of the energy surface over the sub-space to which the input weights belong.

5.1 Training Polynomial Input Functions

Using the definition for δ and equation (2.5) we can write the change of network energy resulting from the change in any input polynomial coefficient (weight) as:

$$\frac{\partial E}{\partial w_{jik}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{jik}} = -\delta_j \frac{\partial net_j}{\partial w_{jik}} = -\delta_j \frac{\partial}{\partial w_{jik}} \sum_s \sum_r w_{jsr} o_s^r = -\delta_j o_i^k \quad (5.3)$$

also from equation (2.7) we can write:

$$\frac{\partial E}{\partial o_j} = \sum_k \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial o_j} = \sum_k \delta_k \frac{\partial}{\partial o_j} \sum_r p_{kr} = \sum_k \delta_k \frac{\partial p_{kj}}{\partial o_j} \quad \text{where} \quad \frac{\partial p_{kj}}{\partial o_j} = \sum_r w_{kjr} o_j^{r-1} r$$

then from equation (2.6):

$$\delta_j = -A'(net_j) \sum_k \delta_k \frac{\partial p_{kj}}{\partial o_j} \quad (5.4)$$

So it can be seen, that once the deltas have been computed by backpropagation, using 5.4, which is just a generalisation of 2.8 then the gradient of the energy surface with respect to the input polynomial coefficients is easily found.

5.2 Training Activation Function Parameters

Let the output of node j be $o_j = A(net_j, \alpha_{1j}, \dots, \alpha_{mj})$ where f is the activation function and α is some parameter which we adjust to minimise the network energy.

Now $\frac{\partial E}{\partial \alpha_{kj}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \alpha_{kj}}$ and from 2.6 we have $\frac{\partial E}{\partial o_j} = -\delta_j / \frac{\partial A}{\partial net_j}$ and as $\frac{\partial o_j}{\partial \alpha_{kj}} = \frac{\partial A}{\partial \alpha_{kj}}$

we can write $\frac{\partial E}{\partial \alpha_{kj}} = -\delta_j \left(\frac{\partial A}{\partial \alpha_{kj}} / \frac{\partial A}{\partial net_j} \right)$, also if $A() = A(q_j(net_j, \alpha_{1j}, \dots, \alpha_{mj}))$ then

$$\frac{\partial E}{\partial \alpha_{kj}} = -\delta_j \left(\frac{\partial q_j}{\partial \alpha_{kj}} / \frac{\partial q_j}{\partial net_j} \right) \quad (5.5)$$

We see that the gradient of the energy surface with respect to the activation function parameters is easily found once the deltas have been computed.

For example if $q_j = \sum_k c_{jk} net_j^k \quad k \in [1, m]$ then $\frac{\partial q_j}{\partial c_{kj}} = net_j^k$ and $\frac{\partial q_j}{\partial net_j} = \sum_{s=1}^m s c_{sj} net_j^{s-1}$

so from 5.5 $\frac{\partial E}{\partial c_{kj}} = -\delta_j / \left(\sum_s s c_{sj} net_j^{s-k-1} \right)$

6. Concluding Remarks

This account has described a global optimisation technique and tried to convey the power of such an approach when used to training ANNs. Although local minima will always result in the consumption of computer resources during training, they need no long be terminal. Once the problem of local minima can be managed, a variety of neural models become tenable. The resulting network architectures are often much simpler than their



classical counterpart and interpretation of a network's internal representation that much more tractable.

References

1. Chao J. How to Find Global Minima in Finite Times of Search
IJCNN 1991 2:1079-1084
2. Cybenko G, Approximations by Superpositions of a Sigmoidal Function.
Mathematics of Control, Signal and Systems 1989; 2:303-314
3. Jacobs RA. Adaptive Mixtures of Local Experts
Neural Computation 1991 3:79-97
4. Dawson M, Schopflocher D. Modifying the Generalised Delta Rule to Train
Networks of Non-monotonic Processors for Pattern Classification.
Connection Science. 1992 4:19-31
- 5a. W.H Press. Numerical Recipes Cambridge 1990 332-345
- 5b. Hertz J. Introduction to the Theory of Neural Computation
Addison-Wesley 1991 pg 109
- 5c. Haykin s. Neural Networks
Macmillan 1994 217-219
6. McClelland and Rumelhart. Parallel Distributed Processing MIT Press 1986
7. Brown R. Gray Layer Technology: Incorporating A Priori Knowledge into
Feedforward Artificial Neural Networks
IEEE 1992 1:806-811
8. Rissanen J. Modeling by Shortest Data Description
Automatica 1978 14:465-471
9. Russel R. Pruning Algorithms - A Survey
IEEE Transactions on Neural Networks 1993 4:740-747
10. Bishop CM. Curvature Driven Smoothing in Backpropagation Neural Networks
CLM-P-880 AEA Technology, Culham Laboratory, Abingdon, England.
11. Piazza F. Neural Network Complexity Reduction Using Adaptive Polynomial
Activation Functions.
ICANN 1993 452-455