Tina Memo No. 1989-001
Internal Report

Developments to the Back-Propagation Learning Algorithm.

N.A.Thacker, John E. W. Mayhew.

Last updated
20 / 11 / 2009

Imaging Science and Biomedical Engineering Division,
Medical School, University of Manchester,
Stopford Building, Oxford Road,
Manchester, M13 9PT.

Developments to the Back-Propagation Learning Algorithm.

Neil A. Thacker, John E. W. Mayhew.

AI Vision Research Unit, University of Sheffield

The original back-propagation methods were plagued with variable parameters which affected both the convergence properties of the training and the generalisation abilities of the resulting network. These parameters presented many difficulties when attempting to use these networks to solve particular mapping problems. A combination of established numerical minimisation methods (Polak-Ribiere Conjugate gradient descent) with a novel idea for stepping out of local minima has produced a network minimisation package which is self optimising and has essentially no free parameters.

Keywords: neural nets, mapping, training.

Introduction

Back-propagation is a training algorithm used to make a layered network learn a functional mapping between a set of input patterns and their respective outputs. The networks take the form of a set of input units connected by wires to an arbitrary number of hidden units, which are connected in turn to output units. It is assumed that each unit in the network is a simple processing element. These produce an output which is simply a differentiable function of the sum of the weighted inputs from the nodes to which it is connected.

In principle any unique mapping can be learned, in practice however there are several problems which have to be addressed first. The first problem is that the original gradient descent method (and its modifications) have a number of free parameters which affect the convergence of particular networks differently. The best parameter values for a given network mapping cannot be predicted in advance and slight error on the choice of these parameters can cause problems with convergence. Also, no methods yet exist for predicting the optimal network architecture for a given mapping problem. These two factors make the application of such networks to even the simplest problem difficult, because if a particular network does not converge we do not know which is responsible. Also, there is no guarantee that choice of the correct parameters and architectures will generate a particular mapping. This is because the back-propagation algorithm is a minimisation algorithm which is subject to the same local minima difficulties as any other such method.

In view of these problems it might be difficult to see why back-propagation networks have almost single-handedly revived interest in network research (for a review see Lippman 1987). The reason is that despite these difficulties these networks have still been shown to be capable of generating functional mappings which have some interesting properties. Most importantly, smooth interpolation gives the network a limited amount of generalisation ability. As a specific example, a network trained on the parity of 20 of the possible 32 five bit binary numbers can often predict correctly the parity of the remaining 12 binary numbers. Optimistic researchers have suggested that these networks are generating algorithms within the network in order to solve the problem. Thus provided the input problem is solvable there is a high probabiliy that such an algorithm will be logically sound and thus automatically have generalisation abilities. Unfortunately few of the problems to which such networks are applied are as constrained as five bit parity and sensible predictive behaviour has to be tested for.

Another use of these networks is in solving the inverse kinematic problem (Elsey 1988). For years researchers devoted time to finding the optimal way of converting from normal cartesian co-ordinates into the configuration spac of robot joint angles. The forward calculation from joints to space is trivial, given an adequate model of the robot, but the inverse kinematic problem was definitely not. However, it is such mappings as these that back-propagation networks learn with ease. Many examples of the forward kinematics and the resulting robot arm position are used as training examples for the network. Once trained these networks provide quicker solutions than the conventional computational approaches. This elegant solution suggests a lot of similar applications of networks to other complex inverse mapping problems, for instance "shape from " modules. Hence, back-propagation networks must be considered as a useful tool for solving many computational problems.

Network modules are likely to grow larger as the problems they are used to solve become more complicated. It is thus important to try to improve the training algorithms and see if any better methods can be found which have fewer unknown factors and are less sensitive to the starting conditions in the untrained network (ie robust). We may then be able to develop systematic methods for the application of these networks to specific classes of problems.

Network structure.

The general network structure (figure 1) and the back-propagation algorithm (Rumelhart Hinton Williams 1986) are now well established in the literature. A vector of input components I_i is fed directly to the output o_i of the input layer i of nodes and the activity at subsequent nodes is calculated from the equations.

$$I_j \;=\; \sum_i w_{ij}.o_i$$

and

$$o_j \;=\; f(I_j)$$

the usual choice for the transfer function $f(x)$ is the ogive (or sigmoid)

$$f(x) \;=\; \frac{1}{1+exp(-x)}$$

Where w_{ij} is the connection strength of the wire between nodes i and j. The input function often also has a bias weight term, but this can be accommodated within this mathematical model by connecting a bias node (with fixed output and no inputs) to node j.

The activity at the output layer is calculated by computing the output from each layer of nodes starting from the input values and working towards the output. Training then proceeds by changing the weight values so that the next time this particular pattern is presented the output generated is more like (by some pre-determined metric) the required result. This training proceedure can be organised in several ways which will be discussed.

Back-propagation by gradient descent.

The training of the network can be envisaged as the minimisation of an error function E_n

$$E_n \;=\; \sum_k \;(o_k - t_{nk})^2$$

where t_{nk} is the required output from node k for input pattern n. There is no intrinsic significance in most cases for choosing this norm, although this (least square error) is the best quantity to use if the training data is expected to have uniform gaussian errors. This equation allows the calculation of the quantity $\partial E_n/\partial w_{ij}$ by repeated application of the chain rule.

$$\partial E_n/\partial w_{ij} \;=\; \partial o_j/\partial w_{ij}.\partial E_n/\partial o_j$$

$$\partial E_n/\partial o_j \;=\; \sum_k \partial E_n/\partial w_{jk}.w_{kj}/o_j$$

For the transfer function given above this derivative calculation is of a particularly simple form. Most of the computation necessary is done during the forward pass so that we almost get the derivative information free (McClelland Rumelhart 1986). Calculation is done starting from the output layer and working backwards towards the input layer hence the name back-propagation. Training can then be achieved by changing all of the weights in the network by a small amount in this direction.

$$\Delta w_{ijn} \;=\; \alpha \partial E_n/\partial w_{ij}$$

Where *alpha* is known as the learning rate and simply determines how far each minimisation step attempts to move in the downward direction.

At first this appears a sensible training stratergy as the data can be presented in any order and there is only one free parameter α. However, this one free parameter is not easy to choose as too small a value will result in a long convergence time and too large a value will result in unstable learning. Further the optimal value will vary during minimisation thus the method cannot be expected to reach the minimum in all cases. Worse still is the fact that there is no real way of monitoring the overall learning state of the network. Although the network will improve its ability to reproduce each particular output at each step there is no reason to expect that this improvement will not be undone by some future learning step. In fact as a minimisation method this is almost certainly the slowest and least robust of all. Some of these problems are however solvable as will be shown below.

Back-propagation with momentum terms.

The first thing to do is to define the error function so that it encompasses all of the available data. Then the total error function is simply redefined as

$$E(\tau) \;=\; \sum_n E_n$$

with

$$\partial E(\tau)/\partial w_{ij} \;=\; \sum_n \partial E_n/\partial w_{ij}$$

Now we can use this quantity after each epoch τ of training (one epoch defined over the whole data set) as a measure of the success of the training. Also we can modify the learning algorithm to include an extra term

$$\Delta w_{ij}(\tau) \;=\; \alpha \partial E(\tau)/\partial w_{ij} + \eta \Delta w_{ij}(\tau - 1)$$

η is generally called the momentum term and does two things. First it smooths out local irregularities in the minimisation function allowing the gradient descent to follow a consistent path. Secondly it allows the minimisation process to increase in speed when there are long periods of identical gradient evaluations.

These properties make this method an order of magnitude faster than the previous one. Despite these improvements, it is this form, with two unknown free parameters, which is unreliable and requires much effort to obtain repeatable minimisation for a particular problem. Yet it is this form of back-propagation which is the base learning algorithm for much research.

Back-propagation with a simple feedback loop.

The next thing that can be tried to increase the speed of the minimisation process is to automate the selection of learning rate using feedback information. Before we do this it is best to reformulate the calculation of the weight change.

$$v_{ij}(\tau) \;=\; \partial E(\tau)/\partial w_{ij} + \beta v_{ij}(\tau - 1)$$

$$\Delta w_{ij}(\tau) \;=\; \alpha v_{ij}$$

β must be chosen in the range 0-0.5 if the previous direction calculations are not to have greater significance than the current derivative in the calculation of the new minimisation direction $v_{ij}(\tau)$. This value is thus easier to specify than the *eta* term and a value of 0.3 is normally used. These changes give α better control of the learning rate and make the new term β responsible solely for smoothing the direction of descent. Also, It should be noted that this formulation is directly comparable to that of the conjugate gradient method which will be described later.

Using the derivative information we can explicitly compute the expected reduction in the error function.

$$\Delta E_e(\tau) \;=\; \sum_{ij} \partial E(\tau)/\partial w_{ij}.\Delta w_{ij}(\tau)$$

Where the summation subscript ij implies all wires in the network. This can then be compared with the achieved value, if they are almost equal (within a few percent) then the linear prediction from the gradient is accurate so we must be on a downward slope relatively free from second order terms and we can therefore increase α with safety. If however the prediction does not agree with the results we must proceed more cautiously and the learning rate must be reduced.

An algorithm of this type was successfully implemented and was found to be significantly faster than the previous methods and also more stable. The method is not entirely robust as occasionally the feedback mechanism cannot respond quickly enough to local changes in the gradient of the error function. This could however be remedied by storing the last successfully computed point as will be described in the conjugate gradient descent method.

Other researchers have implemented feedback modifications based on the acheived minimisation at each step (Vogl et al. 1988). These can be implemented as simple extensions to the original algorithms and result in a similar increase in minimisation speed.

Back-propagation using conjugate gradient descent methods.

In order to remove the final free parameter from the minimisation process we must find some way of determining the optimal value for β locally. In fact an existing minimisation method, known as conjugate gradient descent,

does just this and in a way that can be shown to produce an efficient search for the minimum. The algorithm assumes that we wish to take steps in an approximate downhill direction $v_{ij}(\tau)$ (as above) such that each step is always conjugate to any of the previous steps. This removes the oscillatory behaviour of pure gradient descent methods and is superlinearly convergent in the region of a minimum. It can be shown that such directions can be constructed iteratively from the error derivative provided that each derivative evaluation is done at the minimum along the direction of the previous conjugate direction (for details see **Numerical Recipes in C. 1988**) with

$$\beta(\tau) \;=\; \frac{\sum_{ij}(\partial E(\tau)/\partial w_{ij} - \partial E(\tau-1)/\partial w_{ij}).\partial E(\tau)/\partial w_{ij}}{\sum_{ij} \partial E(\tau-1)/\partial w_{ij}.\partial E(\tau-1)/\partial w_{ij}}$$

As successive derivative directions are generally orthogonal, this is no more than simply the square of the magnitude of the new derivative divided by the square of the old. This is the Polak-Ribiere implementation of the algorithm which allows the direction to revert to the downhill direction when in difficulty. Implementation requires a method of finding the $\alpha(t)$ which gives the minimum of the error function in this direction and in doing so also determines its optimal value for each step.

This linear search for the minimum has to be done in two stages, first the minimum must be bracketed and then the position of the minimum needs to be determined to a specified accuracy within this bracket. The method we adopted for bracketing the minimum was a down hill step search, combined with parabolic extrapolation once three or more evaluations had been done. The minimum was then located using a golden section search algorithm (**Numerical Recipes in C. 1988**), one of the simplest and most robust methods for this task (given that the error function is not likely to be well behaved). The number of function evaluations in each line minimisation can be regulated using the required fractional minimisation at each step without any loss of robustness.

The error function evaluation was done as a current position plus an offset in the conjugate direction so that it was possible to recover from problematic function evaluations. This feature was required for the implimentation of this method and was a very important step in making the minimisation method robust. There is no reason why this technique should not be used in the previous methods to increase their robustness also, as Vogel et al. already seem to have done.

This algorithm is the most robust of all the methods mentioned so far. It has essentially no free parameters as they are all either determined at each step or specified by the floating point accuracy of the implementation. However, the efficient search through the minimisation space is offset by the increased number of function evaluations so that the overall minimisation time is approximately the same as for the methods with feedback. Also, its very robustness now produces a new problem on encountering local minima which has to be overcome.

Problems with local minima.

Knowledge of the expected errors on the training data should be used to determine a sensible value for the desired error function and once below this there can be little point in attempting to leave local minima. Indeed continuing minimisation may be entirely the wrong thing to attempt as any local minimum may embody a logically viable solution. However, if considerable computing time has been invested in training and a local minimum is found, such that the value of the error function is not satisfactory, then it would make sense to try to find a way out of the local minimum so that learning can continue, rather than having to restart the training from a different (probably random) position.

The conjugate gradient minimisation algorithm described above is extremely robust and can be implemented in such a way that it never fails to find a minimum. However, there is no guarantee that the minimum found will be the 'true' global minimum or even close to this. The less robust gradient descent methods can sometimes 'tunnel' out of these minima, but the conjugate gradient descent method will be stuck without an algorithm to step out of them. Much work has already been done on this general minimisation problem but we would like to point out that in the case of network training by back-propagation there is a method available which requires only a slight modification of the existing algorithm.

It was mentioned above that the choice of norm for the error function is in most respects arbitrary. In fact we can choose any continuous difference metric we like to quantify the difference between the obtained and the required output. Specifically we can use any metric of the form

$$E(\tau) \;=\; \sum_n h(\sum_k g(o_k - t_{nk}))$$

which will force $o_k = t_{nk}$ after training. We can expect different forms of this error function to have different local minima in many cases and we will generally expect only the global minimum to be reproduceably coincident. We can use this as a basis for a controlled method of walking out of local minima.

When a local minimum is encountered the value of the error function is stored and a different error function is used until the evaluation of the old function is less than the stored value. By definition the new position must be outside the region of the previous local minimum and we can revert to the original error function if we wish.

The preferred error function for this purpose is

$$E'(\tau) \;=\; \sum_n (\sum_k (o_k - t_{nk})^2)^2$$

as this leads to a simple extension to the existing derivative calculation

$$\partial E'(\tau)/\partial wij \;=\; \sum_n \sum_k (o_k - t_{nk})^2 . \partial E(\tau)_n / \partial w_{ij}$$

This was motivated by the observation that local minima were often the result of a few badly learned mappings and places emphasis on their contribution to the error function and its derivative. In most minimisation problems encountered this method has been found sufficient to escape local minima, allowing minimisation to continue using the original error function.

Discussion.

Considerable research has been devoted to improving convergence speed and robustness of the back-propagation training algorithm. Many more questions could be asked about the benefits of particular transfer functions and architectures, but are there more fundamental problem we should be addressing?

The fundamental aim of AI research must be to develop systems which are capable of learning so that they are not limited to the knowledge available at the time of construction. Such systems may then be capable of extending their knowledge and eventually exhibiting truly intelligent behaviour. In this respect neural networks is currently one of the few research fields which may to be able to support artificial intelligence.

However, if neural networks are to provide a full solution some key issues still need to be answered. Flexible architectures need to be developed which can cope with the problems of noisy data and can efficiently combine multiple sources of information. Integration of the temporal dimension into network function will also be important for a large class of applications (McCulloch 1988). Algorithms need to be developed which train and modify these networks automatically in a robust manner (Linsker 1988). Or at least allow a working definition of efficiency by which to optimise the network architecture for a given problem.

Our belief is that a suitable definition of efficiency can be obtained in terms of the ability of networks to generalise. This is a direct test of the logic embodied in a given architecture and can be combined with the principle of Occam's razor to provide a principled way of finding appropriate architectures: starting with the simplest and only increase the complexity if it is found to give improvement in generalisation performance. This is now becoming a central theme in our own continuing research with use of a back-propagation network for medical diagnosis.

Networks would clearly be more powerful if they generated their own abstract representations during training and these could somehow be made explicit. Work is needed to develop methods for analysing back-propagation networks to determine if they are generating these representations (Lehky Sejnowski 1988 & Sejnowski Rosenberg 1987).

Conclusion.

The speed and robustness of convergence for different variations of the back-propagation minimisation algorithm have been explored. It has been shown that these methods all conform to the same basic mathematical minimisation model but with different numbers of free parameters. The fastest and most robust of these methods has been based on conjugate gradient descent which has essentially no free parameters, each parameter being optimally determined at each epoch during training. We feel that this method is therefore the best of what is currently available (even though the simpler methods may well be adequate for most purposes). We have also developed a simple algorithm for stepping out of local minima when the current minimum is not considered satisfactory. These algorithms have been implemented in C on a Sun workstation and these routines should now provide a solid base from which to explore the application of back propagation neural networks to particular problems.

We argue that existing back-propagation network architectures are a useful tool which should be considered for the solution of certain types of problem and that improvements in training algorithms will help in this. However, the realisation of the potential of these networks, in the field of machine learning, will require fundamental developments to their architectures.

References

D.E. Rumelhart G.E. Hinton R.J. Williams, Learning Representations by Back-propagating errors. Nature 323,533-538,1986.

J.L. McClelland D.E. Rumelhart, Parallel Distributed Processing. MIT press Vol 1 322-328 1986.

R.P.Lippman, An Introduction to Computing with Neural Nets. IEEE ASSP Magasine April 1987.

T.P.Vogl J.K.Mangis A.K.Rigler W.T.Zink D.L.Alkon. Accelerating the Convergence of the Back-propagation Method. Biol. Cybern. 59,257-263,1988.

R.K. Elsey, A Learning Architecture for Control Based on Back-Propagation Neural Networks. Rockwell International Science Center 1988.

W.H.Press B.P.Flannery S.A.Teukolsky W.T.Vetterling Numerical Recipes in C. Cambridge University Press 1988.

N. McCulloch, Methods of Incorporating Time into Artificial Neural Networks. RIPRREF/1000/30 1988.

R. Linsker, Self-Organisation in a Perceptual Network. IEEE Computer 105-118 March 1988.

S.R.Lehky T.J.Sejnowski, Network model of shape from shading: neural function arises from both receptive and projective fields. Nature Vol 333,452-454 June 1988.

T.J.Sejnowski C.R.Rosenberg, Parallel networks that learn to pronounce english text. Complex Systems 1(1),145-168,1987.