On Training Of Feed Forward Neural Networks

Luma.N.Mohammed .Tawfiq*

Date of acceptance 6/9/2005

Abstract:

In this paper we describe several different training algorithms for feed forward neural networks(FFNN). In all of these algorithms we use the gradient of the performance function, energy function, to determine how to adjust the weights such that the performance function is minimized, where the back propagation algorithm has been used to increase the speed of training. The above algorithms have a variety of different computation and thus different type of form of search direction and storage requirements, however non of the above algorithms has a global properties which suited to all problems.

INTRODUCTION:

Back propagation (BP) process can train multilayer FFNN's. With differentiable transfer functions, to perform a function approximation to continuous function f $\in$ Rn, pattern association and pattern classification. The term of back propagation to the process by which derivatives of network error with respect to network weights and biases, can be computed. This process can be used with a number of different optimization strategies.

There are two different ways in which BP algorithms can be implemented; incremental mode and batch mode .All of algorithms, in this paper, operate in the batch mode and are invoked using certain type of training.

1.Variable Learning Rate

With standard gradient descent, the

learning rate is held constant through out training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm become unstable. If the learning rate is too small, the algorithm will take too long to converge. Our numerical results shows that it is not practical to determine the optimal setting for the learning rate before training and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface.

We now describe in some detail one-dimensional search procedure that is guaranteed to find a learning rate satisfying the strong Wolfe conditions (1). As before, we assume that ρ is a search direction and that f is bounded below along the direction ρ. The algorithm has two stages. The first stage begins with a trial estimate $\alpha 1$, and keeps increasing it until it finds either an acceptable learning rate or an interval of desired learning rates. In the latter case, the second stage is invoked by calling a function called zoom (Zoom Algorithm), which successively decreases the size of the interval until an acceptable learning rates is identified. Now we introduce Strong Wolfe Conditions:

$$f(w_k + \alpha_k\rho_k) \le f(w_k) + 10^{-4}\alpha_k\nabla f_k^T \rho_k \ldots (1a)$$

$$\left|\nabla f(w_k + \alpha_k\rho_k)\, \rho_k\right| \le 0.1\left|\nabla f_k^T \rho_k\right| \quad \ldots. (1b)$$

Variable Learning Rate Algorithm:

Set $\alpha_0 \leftarrow 0$, choose $\alpha_1 > 0$ and $\alpha_{max.}$;

$i \leftarrow 1$;

*College of Education Ibn Al-Haitham, Baghdad University

repeat

Evaluate $\phi(\alpha_i)$;

If $\phi(\alpha_i) > \phi(0) + 10^{-4}\alpha_i\phi'(0)$ or $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ and $i > 1]$

$\alpha_* \leftarrow$ zoom (α_{i-1}, α_i) and stop ;

Evaluate $\phi'(\alpha_i)$;

If $|\phi'(\alpha_i)| \leq -0.1\phi'(0)$

Set $\alpha_* \leftarrow \alpha_i$ and stop ;

If $\phi'(\alpha_i) \geq 0$

Set $\alpha_* \leftarrow$ zoom(α_{i-1}, α_i) an stop ;

Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{max})$

$i \leftarrow i + 1$,

end (repeat).

Note that, the sequence of trial learning rates $\{\alpha i\}$ is monotonically increasing, but that the order of the arguments supplied to the zoom function may vary. The procedure uses the knowledge that the interval $(\alpha i - \square\square, \alpha i)$ contains learning rate satisfying the strong Wolfe conditions if one of the following three conditions is satisfied:

(i) αi violates the sufficient decrease condition ;

(ii) $\phi(\alpha i) \geq \phi(\alpha i - 1)$;

(iii) $\phi'(\alpha i) \geq 0$.

The last step of the algorithm performs extrapolation to find the next trial value $\alpha i+1$. To implement this step, we can use approaches like the interpolation procedures above, or we can simply set $\alpha i+1$ to some constant multiple of $\alpha\square$.

We now specify the function zoom, which will requires a little explanation. The order of its input arguments is such that each call has the form zoom $(\alpha_{Lo}, \alpha_{hi})$, where:

a) The interval bounded by αLo and αhi contains learning rates that satisfy the strong Wolfe conditions;

b) αLo is among all learning rates generated so far and satisfying the sufficient decrease condition, the one giving the smallest function value; and

c) αhi is chosen so that $\phi'(\alpha lo)(\alpha hi - \alpha 10) < 0$.

Each iteration of zoom generates an iterate αj between αLo and αhi, and then replaces one of these end points by αj in such a way that the properties (a), (b) and (c) continue to hold.

Zoom Algorithm :

Repeat

Interpolate (using quadratic, cubic ، or bisection) to find a trial learning rate αj between αlo and αhi ;

Evaluate $\phi(\alpha j)$;

If $\phi(\alpha j) > \phi(0) + 10 - 4\phi'(0)$ or $\phi(\alpha j) \geq \phi(\alpha lo)$

$\alpha_{hi} \leftarrow \alpha_j$;

else

evaluate $\phi'(\alpha_j)$;

if $|\phi'(\alpha_j)| \leq -0.1\phi'(0)$

set $\alpha_* \leftarrow \alpha_j$ and stop ;

if $\phi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \geq 0$

$\alpha_{hi} \leftarrow \alpha_{lo}$;

$\alpha_{lo} \leftarrow \alpha_j$;

end (repeat).

If the new estimate α_j happens to satisfy the strong Wolfe conditions, then Zoom has served its purpose of identifying such a point, so it terminates with $\alpha^* = \alpha j$. Otherwise, if αj satisfies the sufficient decrease condition and has a lower function value than αLo, then we set $\alpha_{Lo} \leftarrow \alpha_j$ to maintain condition (b). If this results in a

violation of condition (c), we remedy the situation by setting αhi to the old value of αLo.

2. Resilient Backpropagation (trainrp):

Multilayer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called "squashing" functions, since they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slope must approach zero as the input gets large. This causes a problem when using steepest descent to train a multilayer network with sigmoid functions, since the gradient can have a very small magnitude; and therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

The purpose of the resilient back propagation (Rprop) training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor delt_inc whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by a factor delt_dec whenever the derivative with respect that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating the weight change will be reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change will be increased.

3.BFGS Algorithm (TRAINBFG);

The basic step of this method is $x_{k+1} = x_k - A_k^{-1} g_k$

where A_k is the Hessian matrix (second derivatives)[1] of the performance index at the current values of the weights and biases and g_k is the gradient of the error surface at w(k). This method often converges faster than conjugate gradient methods. Unfortunately, it is complex and expensive to compute the Hessian matrix for FFNN. There is a class of algorithms that is based on Newton's method, but which doesn't require calculation of second derivatives. They update an approximate Hessian matrix at each iteration of the algorithm. The update is computed as a function of the gradient. The BFGS method that has been most successful in published studies is the Broyden, Fletcher, Goldfarb, and Shanno update. This algorithm has been implemented in the trainbfg routine.

For a very large ANN it may be better to use resilient BP or one of the CG algorithms. For smaller ANN, however, BFGS algorithm can be used as an efficient training function.

4. One Step Secant Algorithm (TRAINOSS);

Since the BFGS algorithm requires more storage and computation in each iteration than the conjugate gradient algorithms, there is need for a secant approximation with smaller storage and computation requirements. The one step secant (OSS) method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms. This algorithm does not store the complete Hessian matrix; it assumes that at each iteration, the previous Hessian was the identity matrix. This has the additional advantage that the new search direction can be calculated without computing a matrix inverse. This algorithm requires less storage and computation per epoch than the BFGS algorithm. It requires

slightly more storage and computation per epoch than the conjugate gradient algorithms. It can be considered a compromise between full quasi-Newton algorithms and conjugate gradient algorithms.

5. LM Algorithm (TRAINLM):

The Levenberg-Marquardt (LM) algorithm was designed to approach second order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated as $H = J^T J$ and the gradient can be computed as $g = J^T e$, where J is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors. The LM algorithm uses this approximation to the Hessian matrix in the following Newton update: $w(k+1) = w(k) - [J^T J + \mu I]^{-1} J^T e$.

When the scalar $\mu = 0$, this is just Newton's method. When μ is large, this becomes gradient descent with a small step size.

The training parameters for trainlm are epochs, show, goal, time, min_grad, max_fail, mu, mu_dec, mu_inc, mu_max, mem_reduc. The parameter mu is the initial value for μ. This value is multiplied by mu_dec whenever the performance function is reduced by a step. It is multiplied by mu_inc whenever a step would increase the performance function. If mu becomes larger than mu_max, the algorithm is stopped. The parameter mem_reduc is used to control the amount of memory used by the algorithm.

6. CG Algorithms (TRAINCG):

The conjugate gradient (CG) algorithms perform a search along conjugate directions, which produces generally faster convergence than gradient descent directions [Hagan and Beale, 1996]. The CG algorithms start out by searching in the gradient descent direction (negative of the gradient) on the first iteration ،$\rho 0 = -g0$. Then the next search direction is determined so that it is conjugate to previous search directions, that is:

$w(k+1) = w(k) + \alpha_k \rho_k$. Where $\rho_k = -g_k + \beta_k \rho_{k-1}$.

The various versions of CG are distinguished by the manner in which the βk is computed.

In this paper, we will present six different variations of CG algorithms with a comparison between them. In most of the training algorithms a learning rate is used to determine the length of the weight update (step size).

In most of the CG algorithms, the step size is adjusted at each iteration. A search is made along the CG direction to determine the step size, which will minimize the performance function along that line search. The CG algorithms that usually used in FFNN as a training algorithm is much faster than variable learning rate back propagation, and are sometimes faster than Resilient BP, although the results will vary from one problem to another.

The general procedure for determining the new search direction is to combine the new gradient descent direction with the previous search direction: $\rho k = -gk + \beta k \rho k-1$.

For Fletcher-Reeves update procedure

(TRAINCGF)[2] : $\beta_k = \dfrac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$

For the Polak - Ribiere update

(TRAINCGP) [3] : $\beta_k = \dfrac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}}$

For the Dixon update **(TRAINCGD)** [4] :

$$\beta_k = \frac{-g_k^T g_k}{\rho_{k-1}^T g_{k-1}}$$

For the Al-Assady and Al-Bayati update **(TRAINCGA)** [5]: $\beta_k = \frac{-g_k^T \Delta g_{k-1}}{\rho_{k-1}^T g_k}$

For the Hestenes - Stiefel update **(TRAINCGH)** [6] : $\beta_k = \frac{g_k^T \Delta g_{k-1}}{\rho_{k-1}^T \Delta g_{k-1}}$

For Reyadh – Luma update **(TRAINCGR)**: $\beta_k = \frac{g_k^T \Delta g_{k-1}}{\rho_{k-1}^T g_{k-1}}$

The training parameters are :epochs, show, goal, time, min-grad, sigma, max-fail, lambda, srchFcn.

The training status will be displayed every show iterations of the algorithm. The other parameters determine when the training is stopped. The training will stop when the number of iterations exceeds an epochs, if the performance function drops below goal, if the magnitude of the gradient is less than mingrad or if the training time is longer than time in seconds. The parameter srchfcn is the name of the line search function and the parameter sigma determines the change in the weight for the second derivative approximation . The parameter lambda regulates the indefiniteness of the derivative.

Remark:

1- For all CG algorithms, the search direction will be periodically reset to the negative of the gradient. The standard reset point occurs when the number of iterations is equal to the number of FFNN parameters (weights and biases).

2- For all CG algorithms, the parameters show and epoch set to 5 and 300, respectively.

Each of the CG algorithms, which we have discussed so far, requires a line search at each iteration. This line search is computationally expensive, since it requires that the ANN response to all training inputs which should be computed several times for each search. But the other hand one can designed an algorithm to avoid the time consuming for performing line search. results for many different problems. It does require the computation of the derivatives (back propagation) in addition to the computation of performance function, but it over comes this limitation by locating the minimum with fewer steps.

7. SPEED AND MEMORY COMPARISON:

It is very difficult to know which training algorithm will be the fastest for a given problem. It will depend on many factors including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the FFNN, the error goal, and whether the FFNN is being used for pattern recognition (discriminant analysis) or function approximation (regression).

In general, on FFNN's which contain up to a few hundred weights the LM algorithm will have the fastest convergence. The trainrp function is the fastest algorithm on pattern recognition problems. However, it does not perform well on function approximation on problems. The CG algorithms, in particular traincgp, seem to perform well over a wide variety of problems, particularly for FFNN's with a large number of weights. The traincgr algorithm is almost as fast as the LM algorithm on function approximation problems (faster for large FFNN's) and

is almost as fast as trainrp on pattern recognition problems. The CG algorithms have relatively modest memory requirements.

The trainbfg performance is similar to that of trainlm. It does not require as much storage as trainlm, but the computation required does increase geometrically with the size of the FFNN, since the equivalent of a matrix inverse must be computed at each iteration. Of the CG algorithms, the traincgd requires the most storage, but usually has the fastest convergence. The traincgh and traincga have easily implemented for large problem.

The variable learning rate algorithm traingdx is usually much slower than the other methods and has about the same storage requirements as trainrp but it can still be useful for some problems. For most situations, we recommend that we try to use the LM algorithm first, if this algorithm requires too much memory, then try traincgp or traincgr or trainbfg algorithm. The following table gives some example convergence times for the various algorithms on one particular regression problem. In this problem a 1-15-1 FFNN's was trained on a data set with 41 input/output pairs until a mean square error performance of 0.009 was obtained. Twenty different test runs were made for each training algorithm to obtain the average numbers shown in the table.

Function	*Technique*	*Time(sec)*	*Epochs*
Trainrp	Rprop.	12.95	185
Traincgh	Hestenes-stiefel CG	27.22	112
Traincgf	Fletcher-Powell CG	18.03	94
Traincgp	Polak-Ribiere CG	18.66	79
Traincgd	Dixon CG	24.53	101
Traincgr	Reyadh-Luma CG	14.98	58
Trainbfg	BFGS Alg.	9.76	38
Trainlm	LM Alg.	2.07	8
Traingdx	Variable learning rate	63.17	124
Traincga	Al-Assady and Al-Bayati CG	7136	54

Now we introduce the following problem. 1-5-1 network, with tansig transfer functions in the hidden layer and a linear transfer function in the output layer, is used to approximate a single period of a sine wave. The following table summarizes the results of training the ANN using nine different training algorithms. Each entry in the table represents 30 different trials, where different random initial weights are used in each trial. In each case, the ANN is trained until the squared error is less than 0.002. The fastest algorithm for this problem is the LM algorithm. On the average ,it is over four times faster than the next fastest algorithm. This is the type of problem for which the LM algorithm is best suited -- a function approximation problem where the network has less than one hundred weights and the approximation must be very accurate.

Algorithm	*Mean.Time(s)*	*Min.Time(s)*	*Max.Time(s)*
LM	1.14	0.65	1.83
BFG	5.22	3.17	14.38
RP	5.67	2.66	17.24
CGF	7.86	3.57	31.23
CGP	8.24	4.07	32.32
OSS	9.64	3.97	59.63
CGR	5.92	2.31	16.47
CGA	27.69	17.21	258.15
CGD	6.09	3.18	23.64
CGH	6.61	2.99	23.65

The performance of the various algorithms can be affected by the accuracy required of the approximation.

8. LIMITATIONS AND CAUTIONS:

The gradient descent algorithm is generally very slow, because it requires small learning rates for stable learning. The momentum variation is usually faster than simple gradient descent, since it allows higher learning rates while maintaining stability, but it is still too slow for many practical applications. These two methods would normally be used only when incremental training is desired. Multi-layered

networks are capable of performing just about any linear or non-linear computation, and can approximate any reasonable smooth function arbitrarily well. Such networks overcome the problems associated with the feed forward and linear networks.

Picking the learning rate for a non-linear network is still an open problem. As with linear networks, a learning rate that is too large leads to unstable learning. Conversely, a learning rate that is too small results in incredibly long training times. Unlike linear networks, there is no easy way of picking a good learning rate for non-linear multilayer networks.

The error surface of a non-linear network is more complex than the error surface of a linear network. The problem is that non-linear transfer function in multilayer networks introduce many local minima in the error surface. Settling in a local minimum may affect the convergence and depending on how close the local minimum is to the global minimum and how low an error is required. In any case, be cautioned that although a multilayer back propagation network with enough neurons can implement just about any function, back propagation will not always find the correct weights for the optimum solution.

References:

1. Yegnanarayana. B. 2000. Artificial Neural Networks, Newdelhi .
2. Fletcher.R.and Reeves.C.M., 1964.Function Minimization by Conjugate Gradients, Computer Journal, (7): 149 –154 .
3. Polak .E .and Ribiere. G. 1969. Note sure La Convergence does methods Directions Conjugate, Rev. Fr. Infr, Rech open, 16-R1, (6):86-97.
4. Dixon. L.G. 1975. Conjugate Gradient algorithms quadratic termination with out linear search, Jor. of Tnst. of Math. and its applications ,(15):124-131.
5. Al - Bayati and Al – Assady N. 1996. Conjugate Gradient Methods, Technical Research Report, NO.1, School of Computer Studies, Leeds University, U. K.,(1):76-83.
6. Hestenes M. R. and Stiefel E. 1952. Methods of Conjugate Gradient for Solving linear System, J. Res. NBS, (49):214-225.

حـول تدريـب الشبكـات العصبيـة الصنـاعيـة

لمى ناجي محمد توفيق*

*أ.م.د. قسم الرياضيات /كلية التربية ابن الهيثم /جامعة بغداد

الخلاصة:

يتضمن البحث مناقشة أنواع مختلفة من خوارزميات تدريب الشبكات العصبيـة ذات التغذيـة التقدمية وفي كل تلك الخوارزميـات استخدمنـا مشتقـة دالـة الطاقـة لتحديد كيفيـة ضبط الأوزان بحيث تصبح دالـة الطاقـة أصغر ما يمكن و لقد استخدمنـا خوارزميـة الانتشار المرتـد لزيـادة سرعـة التدريب. تختلف الخوارزميات أعلاه في حساباتها و لذلك نحصل علـى صيغ متنوعـة في اتجاه التفتيش و الخزن الذي تقتضيـه فقد أثبتت النتـائج العمليـة بـأن أيا من الخوارزميات أعلاه لا تمتلك خواص رئيسية مثل الاستقراريـة و التقـارب و التي تجعلها مناسبة لكل المسائـل .