On the Quasi-Newton Training Method for Feed-Forward Neural Networks

Nikolai Stanevski, Dimiter Tsvetkov

Abstract: *The quasi-Newton training method is the most effective method for feed-forward neural networks with respect to the training precision. This method is well-known and popularly described in the neural networks literature. Nevertheless its implementation contains some difficulties because of the specific shape of the cost function and the large amount of variables. Here we give in sufficient details an example of a* ***program implementation*** *of the quasi-Newton method. This implementation (as a Borland Delphi application) seems to work well with various examples.*

Keywords: *feed forward neural networks, training methods, quasi-Newton method.*

INTRODUCTION

Here we suppose that the reader is acquainted with the basis of the neural networks theory. Feed-forward (**FF**) neural network architecture consists of an input layer, output layer and several hidden layers. Suppose we are given L samples and each sample contains several input signals and several output signals. The problem is to train the network using given data that means to minimize some cost function which usually is the least square function. There are several second order optimization methods that may be used to train the network like conjugate-gradient method, Levenberg-Marquardt method and quasi-Newton method. Remember also the first order methods as steepest descent direction and its modification with momentum addend. All they avoid calculating of the Hessian matrix which calculation is very inconvenient for implementation. It is easy to verify that the quasi-Newton method is the best with respect to accuracy and therefore it is interesting and useful to investigate how it works and how it can be implemented in the practice. The quasi-Newton method is already well implemented in various applications, for example in MATLAB but all they have many hidden parameters and properties. The main aim of the present work is to offer in details a well working variant of quasi-Newton method as a computer implementation in Delphi application.

THE BACK PROPAGATION PROCEDURE

The Back Propagation procedure (**BP**) calculates gradient vector $\mathbf{g}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$. Here follows a brief presentation of **BP**. let $m = 0,1,\ldots,M$ is the layer index where index 0 stands for the input layer and M – for the output one. Denote by $v[m,i]$, $i = 0,1,\ldots,k[m]$, the signals of the m-th layer. It is convenient to use zero indices for the bias signal; then always we have $v[m,0] = 1$. The m-th layer consists of $k[m]$ neurons and the i-th neuron is defined by its weighs $w[m,i,j]$, $i = 1,2,\ldots,k[m]$, $j = 0,1,\ldots,k[m-1]$, and by its transfer function $\varphi_m(u)$. The weight $w[m,i,0]$ stands for the corresponding bias. In this way $k[0]$ is the number of the inputs and $k[M]$ is the number of the outputs. **BP** consists of several steps.

Consider the l-th sample. Then we have $v[0,i] = \mathbf{input}[l,i]$ for $i = 1,2,\ldots,k[0]$, where the array $\mathbf{input}[l,i]$ is used for the set of inputs; for the outputs we use the array $\mathbf{output}[l,i]$, $i = 1,2,\ldots,k[M]$.

BP1. Calculation of signals along the network (*forward step*)

$$h[m,i] = \sum_{j=0}^{k[m-1]} w[m,i,j] * v[m-1,j] \quad (1)$$

$$v[m,i] = \varphi_m\left(h[m,i]\right) \quad (2)$$

for $m = 1,2,\ldots,M$ and $i = 1,2,\ldots,k[m]$. Remember that we always have $v[m,0] = 1$. Here we point out that the cost function for the l-th sample is

$$\mathbf{cf}_l(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{k[M]}\left(v[M,i] - \mathbf{output}[l,i]\right)^2 \quad (3)$$

BP2. Calculation of so-called deltas (backward step) starting from the output M-th layer for which we have

$$\delta[M,i] = \varphi'_M\left(h[M,i]\right) * \left(v[M,i] - \mathbf{output}[l,i]\right), \quad (4)$$

and for $m = M, M-1,\ldots,2$, we calculate

$$\delta[m-1,i] = \varphi'_m\left(h[m-1,i]\right)\sum_{j=1}^{k[m]} w[m,j,i] * \delta[m,j] \quad (5)$$

for $i = 0,1,\ldots,k[m-1]$.

BP3. Calculation of the derivatives for $\mathbf{cf}_l(\mathbf{w})$ with respect to $\mathbf{w}$

$$g_l[m,i,j] = \delta[m,i] * v[m-1,j], \quad (6)$$

for $m = 1,2,\ldots,M$, $i = 1,2,\ldots,k[m]$, $j = 0,1,\ldots,k[m-1]$.

These derivatives are exactly the entries of the gradient

$$\mathbf{g}_l(\mathbf{w}) = \frac{\mathbf{d}}{\mathbf{dw}}\mathbf{cf}_l(\mathbf{w}). \quad (7)$$

Having once the values of $\mathbf{g}_l(\mathbf{w})$ we can update $\mathbf{w}$ with the well-known formula for ***incremental*** learning

$$w_{new}[m,i,j] = w_{old}[m,i,j] - \eta g_l[m,i,j], \quad (8)$$

where η is the learning rate parameter which usually takes value 0.05. In the process of the incremental learning, the steps above are repeated from the first sample to the last one that forms a separate ***epoch***. A large number of epochs is needed to obtain a good network.

In the quasi-Newton method the learning mode is ***batch*** mode instead of the ***incremental*** mode and therefore the cost function is the sum of all the sample cost functions

$$\mathbf{cf}(\mathbf{w}) = \sum_{l=1}^{L}\mathbf{cf}_l(\mathbf{w}). \quad (9)$$

Then obviously

$$\mathbf{g}(\mathbf{w}) = \frac{\mathbf{d}}{\mathbf{dw}}\mathbf{cf}(\mathbf{w}) = \sum_{l=1}^{L}\mathbf{g}_l(\mathbf{w}), \quad (10)$$

and

$$g[m,i,j] = \sum_{l=1}^{L} g_l[m,i,j]. \quad (11)$$

THE QUASI-NEWTON METHOD – layer-by-layer implementation

Now we pay our attention to the ***line-search problem*** that means to solve the following optimization problem. Find $t \geq 0$ such that the scalar function

$$\mathbf{cf}\left(\mathbf{w} - t\,\mathbf{g}\left(\mathbf{w}\right)\right) \tag{12}$$

attains its minimum. This problem can be approximately solved in many ways. Fortunately the quasi-Newton method is not (too much) sensitive to the precision of the solution of the line-search problem. Here one can use straightforward calculations by varying t until some criterion is met (this simple strategy works well).

First of all it is important to know that the **QN** method is an abstract method which gives the ***exact*** solution of the minima problem in the case when the cost function is quadratic with a positively defined Hessian. In this sense **QN** looks better even than the pure Newton-Raphson method. Of course the cost function here is not quadratic but nevertheless **QN** converges well.

We will implement **QN** in a ***layer-by-layer*** mode. Consider the m-th layer (arbitrary chosen, $m = 1,2,\ldots,M$) and let $\mathbf{w}^m$ denotes the vector of the m-th layer weights. Suppose that we have the gradient $\mathbf{g}\left(\mathbf{w}^m\right)$ in a ***vector*** form. Here arises a technical problem to transform the double-indexed array, as the gradient is obtained by **BP**, to one-indexed and conversely. Denote by ω the entry number of $\mathbf{w}^m$. Hereafter for brevity we will omit the upper index m. Let $\mathbf{w}_0$ be the starting value of the weight ω-vector $\mathbf{w}$. (All the weights have to be initialized as small random numbers during the net construction.) Set a starting value of the $\omega \times \omega$-matrix $\mathbf{H}_0 = \mathbf{I}$. **QN** method contains several stages.

QN1. Suppose we have already some current value $\mathbf{w}_n$. Next calculate $\mathbf{g}\left(\mathbf{w}_n\right)$ and find the (approximate) solution t_n of the line-search problem

$$\mathbf{cf}\left(\mathbf{w}_n - t\,\mathbf{H}_n\mathbf{g}\left(\mathbf{w}_n\right)\right) \Rightarrow \min \tag{13}$$

and set

$$\mathbf{w}_{n+1} = \mathbf{w}_n - t_n\,\mathbf{H}_n\mathbf{g}\left(\mathbf{w}_n\right). \tag{14}$$

Calculate now $\mathbf{g}\left(\mathbf{w}_{n+1}\right)$ and put

$$\Delta\mathbf{w}_n = \mathbf{w}_{n+1} - \mathbf{w}_n \quad \text{and} \quad \Delta\mathbf{g}_n = \mathbf{g}\left(\mathbf{w}_{n+1}\right) - \mathbf{g}\left(\mathbf{w}_n\right). \tag{15}$$

The values of the gradients are found by using of **BP**. Now update the $\mathbf{H}$ matrix

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \frac{\Delta\mathbf{w}_n\Delta\mathbf{w}_n^T}{\Delta\mathbf{w}_n^T\Delta\mathbf{g}_n} - \frac{\mathbf{H}_n\Delta\mathbf{g}_n\Delta\mathbf{g}_n^T\mathbf{H}_n}{\Delta\mathbf{g}_n^T\mathbf{H}_n\Delta\mathbf{g}_n}. \tag{16}$$

QN2. Repeat steps **QN1** (until a proper stop criterion is met). The number of repeats is at least ω because the same number of repeats is necessary even in the case of a quadratic cost function. Our experience shows that it is sufficient to choose ω as a number of these repeats.

QN3. Make a pass of steps **QN1** and **QN2** over the layers starting from the output layer to the input one. By analogy this pass can be called an ***epoch***.

QN4. Repeat **QN1-3** steps until the cost function becomes sufficiently small or stop changing itself (do some epochs).

All the $\mathbf{H}$ matrices occur positively defined and therefore the vector $\mathbf{H}_n\mathbf{g}\left(\mathbf{w}_n\right)$ has a positive projection on the gradient $\mathbf{g}\left(\mathbf{w}_n\right)$. In fact $\mathbf{H}_n$ is an approximation to the inverse

Hessian matrix (in the quadratic case $\mathbf{H}_\omega$ coincides with the inverse Hessian). The **QN** method looks straightforward and easy to program. The best fact here is that **QN** converges rapidly. The formulas in method described above corresponds to the so-called Davidon-Fletcher-Powell (**DFP**) method. There exist also other quasi-Newton methods. The Broyden-Fletcher-Goldfarb-Shanno (**BFGS**) method differs from **DFP** only in updating of $\mathbf{H}$:

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \frac{\Delta\mathbf{w}_n \Delta\mathbf{w}_n^T}{\Delta\mathbf{w}_n^T \Delta\mathbf{g}_n} - \frac{\mathbf{H}_n \Delta\mathbf{g}_n \Delta\mathbf{g}_n^T \mathbf{H}_n}{\Delta\mathbf{g}_n^T \mathbf{H}_n \Delta\mathbf{g}_n} + \Delta\mathbf{g}_n^T \mathbf{H}_n \Delta\mathbf{g}_n \mathbf{U}\mathbf{U}^T, \tag{17}$$

where

$$\mathbf{U} = \frac{\Delta\mathbf{w}_n}{\Delta\mathbf{w}_n^T \Delta\mathbf{g}_n} - \frac{\mathbf{H}_n \Delta\mathbf{g}_n}{\Delta\mathbf{g}_n^T \mathbf{H}_n \Delta\mathbf{g}_n}. \tag{18}$$

Separating the implementation of **QN** in the layer-by-layer mode obviously does not disturb the convergence. The numerical complexity of **QN** is $O(\omega^2)$ (in contrast with the other popular methods which complexity is $O(\omega)$) therefore **QN** becomes significant slower than the other methods in the case of very large neural networks. Nevertheless **QN** should be used when the learning precision is of the main importance.

EXAMPLE

In the following example we have **1** input, **2** outputs and **63** samples. Input is the uniform set of points in the segment $[-\pi, \pi]$ by step 0.1. First output is the values of the function $\cos(x)$ and the second output is the values of the function $\sin(x)$. The net has one hidden layer only with **5** neurons with a sigmoid transfer function. The initial error is 31.5. In the following graphics it is shown the error dynamics.

Figure 1. Plots of the output and simulated signals

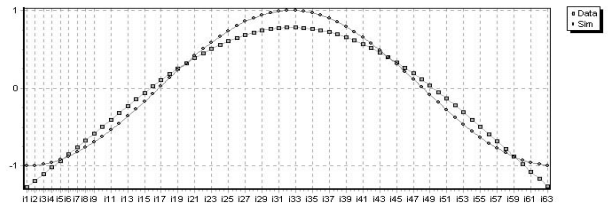
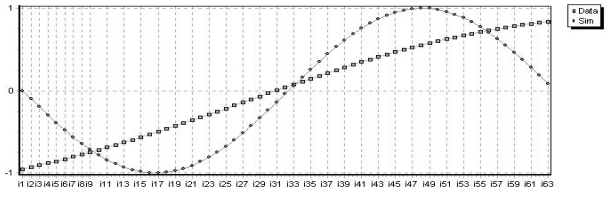
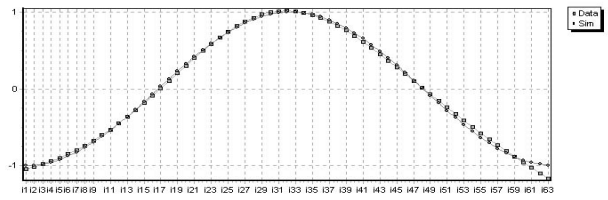
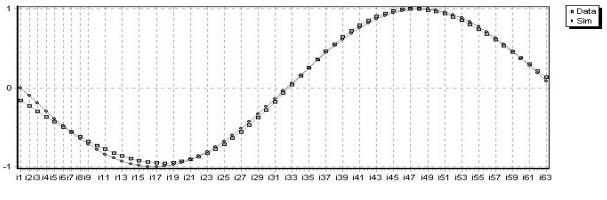
first output –> $\cos(x)$	second output –> $\sin(x)$
after **1** epoch, error=6.024	
	
after **3** epochs, error=0.120	
	

Table 1. Error dynamics for the first 16 epochs

epoch number	error	epoch number	error	epoch number	error
initial	31.500	6	0.090	11	0.079
1	6.024	7	0.087	12	0.076
2	4.128	8	0.086	13	0.075
3	0.120	9	0.084	14	0.073
4	0.093	10	0.082	15	0.070
5	0.091	11	0.081	16	0.066

By this example we see that the error decreases rapidly for the first epochs and then the decreasing becomes slowly. This common phenomenon is observed also for the other examples. Perhaps the reason for this slower error decreasing is hidden in the relatively imperfect solution of the line-search problem. On the other hand the more precise solution of the line-search problem makes the overall algorithm more slower which is the main disadvantage of the quasi-Newton method.

Let now see what is going on with the same example when the quasi-Newton updating of $\mathbf{H}$ matrix is removed and the optimisation relies only on line-search. We need near **250** epochs to reach a visual satisfactory precision.

Figure 2. Plots of the output and simulated signals (only with line-search)

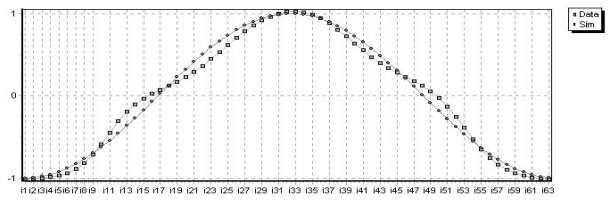
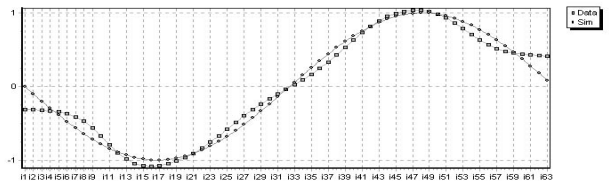
first output –> $\cos(x)$	second output –> $\sin(x)$
after 250 epochs, error=0.573	
	

Table 2. Error dynamics for 170 epochs (only with line-search)

epoch number	error	epoch number	error	epoch number	error
initilal	31.500	60	2.596	120	1.958
10	4.510	70	2.516	130	1.813
20	3.288	80	2.439	140	1.691
30	3.006	90	2.338	150	1.581
40	2.835	100	2.248	160	1.464
50	2.685	110	2.141	170	1.339

Without updating of $\mathbf{H}$, i.e. only with line-search strategy, the convergence becomes very slow (first order optimization method). The latter shows that the **QN** method is very self-consistent and does not admit arbitrary intuitive improvements.

CONCLUSIONS AND FUTURE WORK

Obviously the quasi-Newton method is easy to implement for small neural network architectures. The **QN** method can be explored intensively in many other optimisation problems of practical use. We are going to analyse the Levenberg-Marquardt method which is reported as a very good method both for the precision and the speed of learning process.

REFERENCES

[1] Haykin, S. Neural Networks. Pearson Education, India, 2001.

[2] Luenberger D. Linear and Nonlinear Programming, Addison-Wesley, 1984.

[3] Himmelblau D., Applied Nonlinear Programming, McGraw-Hill Book Company, 1972.

ABOUT THE AUTHORS

Assoc. Prof. Dimiter Petkov Tsvetkov, PhD, Department of Mathematics, National Military University "Vassil Levski", Veliko Tarnovo, Phone: +359 62 600 210, E-mail: dimiter99@yahoo.com.

Nikolai Stoyanov Stanevski, PhD student, Cartographical Institute, Troyan, Phone: +359 670 2 58 59, E-mail: nstanevski@dir.bg.