

Comment on: An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer

B.Ph. van Milligen, V. Tribaldos, J.A. Jiménez and C. Santa Cruz

Abstract—

The present letter analyzes the performance of the neural network training method known as Optimization Layer by Layer [1]. We show, from theoretical considerations, that the amount of work required with OLL-Learning scales as the third power of the network size, compared with the square of the network size for commonly used Conjugate Gradient training algorithms. This theoretical estimate is confirmed through a practical example. Thus, although OLL is shown to function very well for small neural networks (less than about 500 weights per layer), it is slower than CG for large neural networks.

Second, we show that OLL does not always improve on the accuracy that can be obtained with CG. It seems that the final accuracy that can be obtained depends strongly on the initial network weights.

Keywords— Neural Network, Training, OLL, Conjugate Gradient.

I. INTRODUCTION

In Ref. [1], a new method for training Multi-Layer Perceptron (MLP) neural networks was introduced: Optimization Layer by Layer (OLL). The method is based on the linearization of the nonlinear activation functions of the neural network (also known as sigmoids), thus leading to a linear optimization problem for each network layer. The error made by linearizing the activation functions is accounted for by a penalty functional, whose influence is varied (through a parameter μ) to maintain optimum convergence. The neural network is then optimized in an iterative procedure, whereby in each iteration the weights of each layer are optimized by solving a set of linear equations. The algorithm is designed in such a way that the total error of the network decays monotonically.

This method provides a viable alternative to standard gradient-descent optimization methods, and we have found that it is quite powerful. However, claims as to speed and accuracy are grossly exaggerated by the authors and the present comment is meant to put the algorithm's performance in perspective. In doing so, we shall restrict ourselves to the algorithm as proposed by the authors. For numeric comparisons, we shall use both an example given by the authors and an example of our own.

B.Ph. van Milligen, V. Tribaldos and J.A. Jiménez are plasma scientists at the Asociación EURATOM-CIEMAT, Madrid, Spain

C. Santa Cruz does research on neural networks at the Instituto de Ingeniería del Conocimiento, Madrid, Spain

II. SPEED OF OLL-LEARNING

First, we discuss execution speed. The main work of the algorithm is done in *a)* the construction of the matrix (for each layer) that gives the set of equations for the linearized optimization problem and *b)* the actual solution of the equations. As the authors point out, part *b)* can be optimized by using a Cholesky decomposition. As a side-remark, we would like to mention that in general it is necessary to regularize the matrix to avoid numerical problems, e.g. by adding a small number (we used one-hundredth of the smallest absolute value of all matrix elements) to all elements of the diagonal of the matrix. Even using an optimum Cholesky decomposition, the number of floating-point operations required in part *b)* is considerable: about $N^3/3$ for the factorization and about $2N^2H$ for the solution of the set of equations; here N is the total number of weights in the layer to be optimized, and H is the number of target nodes for the same layer.

However, in most of the cases we studied the algorithm spent the largest part of execution time in *a)*, the construction of the matrix. Here we focus on the optimization of the hidden layer as described in Ref. [1], the optimization of the output layer being similar. The number of floating point operations (multiplications and summations) per iteration and per layer (cf. Eq. (36) in Ref. [1]) is $4K$ for each matrix element (K being the number of data in the training set), i.e. about $4N^2K$ operations in total. However, some optimization is possible: first, the product $v_{linh}^k \cdot x_m^k$ can be calculated beforehand, avoiding about $2N^2K$ multiplications; and second, the double symmetry of the matrix can be exploited to reduce the number of calculations by about a factor of 4. In all, that leaves $N^2K/2$ floating point operations. Taking into account that K must be at least 5 to 10 times N in order to avoid overfitting, the total number of floating point operations is at least $(5 \div 10)N^3/2$.

These considerations make the application of OLL-training to large neural networks problematic, since the work required grows as N^3 . On vector computers, the innermost loop in part *a)*, i.e. the loop over K , can be vectorized, implying a considerable speedup (proportional to the vector length), but the outermost loops, representing work of the order of N^2 , remain scalar.

By contrast, the time per iteration required by a Conjugate Gradient (CG) method, although far more difficult to estimate since the time required for a particular iteration depends both on the local difficulty of the problem and on the iteration history of the minimization, scales roughly

with $N_{tot}K$, where N_{tot} is the total number of weights of all network layers (cf. Ref. [2]). Thus, since K must again be at least 5 to 10 times N_{tot} to avoid overfitting, the time per iteration for the CG method increases only quadratically with the network size, whereas the time required for the OLL algorithm increases with the third power.

To see how the calculation time augments with the network size in a practical example, we have applied the technique to the problem of approximating the spatial distribution of the magnetic flux of a stellarator by an MLP-1 neural network having the three spatial coordinates as inputs and the magnetic flux as output. A more detailed description of this type of problem can be found in Ref. [6], although here we are considering a different stellarator (TJ-IU) [7] than the one considered in the cited paper (TJ-II).

We have applied both the OLL algorithm and the CG algorithm to train MLP-1 3: H :1 (3 input nodes, H hidden layers, and 1 output node) neural networks, with H taking the values 8, 31, 127 and 511 (the numbers are chosen to achieve optimum vector performance for the CG algorithm on our Cray J90 machine). In all cases, the number of examples in the training set was $K = 2000$. This was done in order to facilitate comparisons, even though this means that some overfitting will occur at larger values of H .

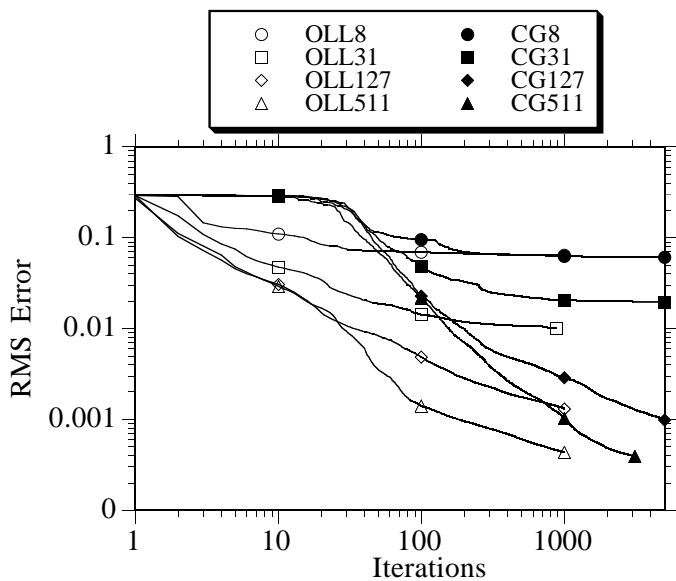


Fig. 1. Comparison of convergence rates for the approximation of a magnetic flux. Calculations are done with both the OLL and the CG method for a 3: H :1 network; the number H takes the values 8, 31, 127 and 511 and is indicated for each curve. The horizontal axis is the iteration count.

From Fig. 1 one observes that the OLL algorithm shows very good convergence rates in terms of iteration count, and especially at small values of H the reached accuracy is higher than the one reached by the CG algorithm for the same initial conditions. This looks very promising, but comparing the CPU time (Fig. 2) another picture emerges: convergence is still fast and good for low values of H , but as H increases things get worse: at $H = 127$, the performance of OLL and CG is equivalent, while at $H = 511$, the CG

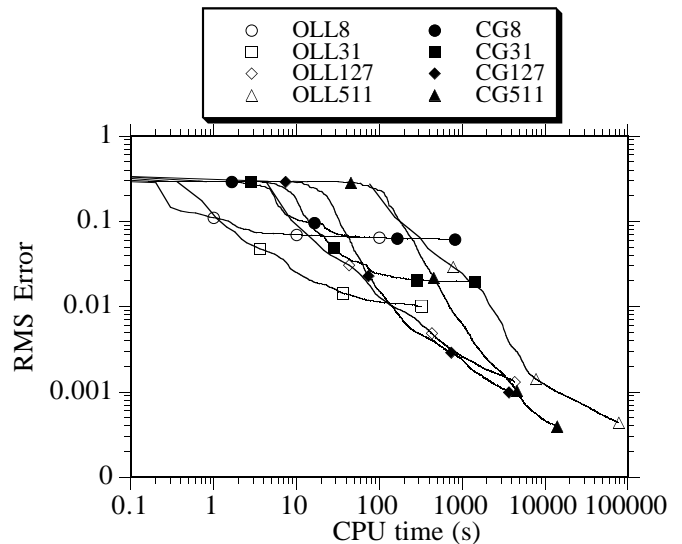


Fig. 2. Comparison of convergence rates for the approximation of a magnetic flux. Calculations are done with both the OLL and the CG method for a 3: H :1 network; the number H takes the values 8, 31, 127 and 511 and is indicated for each curve. Data are the same as in Fig. 1. The horizontal axis is the CPU time in seconds.

algorithm beats the OLL algorithm. These observations are easily understood from the scaling behaviour with N (proportional to H) deduced in section II. It seems that the maximum number of hidden nodes for which the OLL algorithm still presents an advantage over CG is around $H = 127$, corresponding to a total number of network weights of about 500.

III. ACCURACY OF OLL-LEARNING

Claims for the accuracy that can be obtained by OLL-Learning as made in Ref. [1] are truly impressive: e.g. the accuracy obtained for predicting a chaotic Lorenz time series is given as -128 dB for the training set (Section 4B in the cited paper). According to the definition of the Normalized Root Mean Squared Error (NRMSE) by the authors (Eq. (49)), this corresponds to a relative Root Mean Squared (RMS) error,

$$\frac{\langle \|d^k - y^k\|^2 \rangle^{1/2}}{\langle \|d^k - \langle d^k \rangle\|^2 \rangle^{1/2}}$$

of $3.98 \cdot 10^{-7}$, or 0.00004 %. This is difficult to believe, so we decided to repeat the calculation exactly as described by the authors. The Lorenz time series was calculated entirely according to their specifications, with the same initial conditions, using a Runge-Kutta method to integrate the equations with a time step of 0.01 seconds and eliminating the first 3000 points of the calculated series to avoid transients. This time series was then used to train a 5:8:1 (number of input, hidden and output nodes, respectively) neural network, taking five successive samples as the input vector and the next sample as the output vector. We trained the network both using the OLL training scheme and a Conjugate

Gradient method. The Conjugate-Gradient (CG) method we used is the E04UCE subroutine of the Numerical Algorithms Group (NAG) [3], which is essentially identical to the SOL/NPSOL subroutine described in Ref. [4]. This method can be described as a modified quasi-Newton sequential quadratic programming (SQP) method. The computation time required for an iteration in this method can be greatly reduced by supplying the derivatives of the error functional with respect to the network weights to the subroutine. It has been shown by us in earlier publications that these derivatives can be calculated analytically in a very simple manner for MLP networks [2], [5].

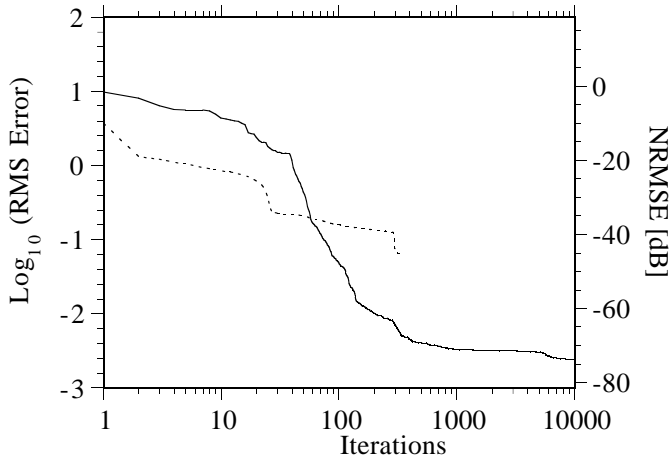


Fig. 3. Comparison of convergence rates for the prediction of a chaotic Lorenz time series. Continuous line: Conjugate Gradient method. Dotted line: OLL.

From Fig. 3, it is at once obvious that (1) the spectacular error levels as cited by the authors are not reached even remotely, and (2) OLL is not better than our Conjugate-Gradient method. The Conjugate Gradient method we used, however, reproduces the error levels obtained by the FRR Conjugate Gradient method that was used in Ref. [1] quite well. We have observed that the OLL training scheme often reaches saturation after a relatively small number of iterations (i.e. the parameter μ begins to grow indefinitely without obtaining any reduction in the error), which is sensitively dependent on the initial network weights. We cut off calculations at that point. In our case, we have used a random number generator, generating numbers in the range $(-1, 1)$, to initialize the network weights. The OLL training curve shown in Fig. 3 is a typical one; the final reconstruction absolute RMS error, $\langle \|d^k - y^k\|^2 \rangle^{1/2}$, either upon reaching saturation or upon reaching the maximum number of iterations, varies for different values of the random number generator seed, but is always of the order of 0.1. The reported value of 0.00004 % in relative error, or $4.7 \cdot 10^{-6}$ in absolute RMS error, was never obtained. Thus, we are led to conclude that the case reported in Ref. [1] is one where, through a very special initial set of network weights, very good reconstruction errors are obtained, but that in general no such performance may be expected. In

this respect, it is to be lamented that the paper does not report how the network is initialized.

IV. CONCLUSIONS

We have shown, from theoretical considerations, that the amount of work required for training a neural network with OLL-Learning scales as the third power of the network size, compared with the square of the network size for commonly used Conjugate Gradient training algorithms. This theoretical estimate has been confirmed through a practical example. Thus, although OLL has been shown to function very well for small neural networks (less than about 500 weights per layer), it is slower than CG for large neural networks.

Second, we have shown that OLL does not always improve on the accuracy that can be obtained with CG. It seems that the final accuracy that can be obtained depends strongly on the initial network weights. This is at once obvious from the fact that OLL, like CG, provides a strictly monotonic descent in reconstruction error. Thus, it is possible that the algorithm converges to a local minimum rather than a global one.

REFERENCES

- [1] S. Ergezinger and E. Thomsen, "An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer", *IEEE Trans. Neural Net.*, **6** (1991) 31.
- [2] B.Ph. van Milligen, V. Tribaldos and J.A. Jiménez, "Neural Network Differential Equation and Plasma Equilibrium Solver", *Phys. Rev. Lett.* **75**, 20 (1995) 3594.
- [3] The Numerical Algorithms Group Limited, *The NAG Fortran Library Manual, Mark 17*, 1st edition (1995), ISBN 1-85206-124-3.
- [4] P.E. Gill, S.J. Hammarling, W. Murray, W. Saunders and M.H. Wright, *User's guide for LSSOL (Version 1.0)*, Department of Operations Research, Stanford University, Technical Report SOL 86-6R.
- [5] B.Ph. van Milligen, J.A. Jiménez and V. Tribaldos, "Solving Differential Equations using a Neural Network; Application to Three-dimensional Plasma Equilibria", Submitted to *J. Comput. Phys.* (1997).
- [6] V. Tribaldos and B.Ph. van Milligen, "Electron Cyclotron Emission Calculations for TJ-II Stellarator", *Nucl. Fusion* **36**, 3 (1996) 283.
- [7] E. Ascasibar, C. Alejandre, J. Alonso, *et al.*, "Initial operation of the TJ-IU torsatron and theoretical studies for the flexible heliac TJ-II" *Plasma Phys. Control. Nucl. Fusion Research*, IAEA-CN-60/A6-1, **1** (1994) 749.