The Development and Analysis of Analytic Method as Alternative for Backpropagation in Large-Scale Multilayer Neural Networks

Mikael Fridenfalk
Department of Game Design
Uppsala University
Visby, Sweden
mikael.fridenfalk@speldesign.uu.se

***Abstract*—This paper presents a least-square based analytic solution of the weights of a multilayer feedforward neural network with a single hidden layer and a sigmoid activation function, which today constitutes the most common type of artificial neural networks. This solution has the potential to be effective for large-scale neural networks with many hidden nodes, where backpropagation is known to be relatively slow. At this stage, more research is required to improve the generalization abilities of the proposed method.**

Keywords-analytic; FNN; large-scale; least square method; neural network; sigmoid

I. Introduction

The artificial neural network constitutes one of the most interesting and popular computational methods in computer science. The most well-known category is the multilayer Feedforward Neural Network, here called FNN, where the weights of the network are estimated by an iterative training method called backpropagation [6]. Although this iterative method is, given the computational power of modern computers, relatively fast for small networks, it is rather slow for large networks [1]. To accelerate the training speed of FNNs, many approaches has been suggested based on the least square method [2]. Although the presentation on the implementation, as well as of the data on the robustness of these methods may be improved, the application of the least square method as such seems to be a promising path to investigate [7].

What we presume to be required for a new method to replace backpropagation in such networks, is not only that it is efficient, but also that it is superior compared to existing methods and is easy to understand and implement. The goal of this paper is therefore to investigate the possibility to find an *analytic* solution for the weights of an FNN, i.e., without any iterations involved, that is easily understood and that may be implemented relatively effortlessly, using a mathematical application such as Matlab [5]. As a brief overview, the analytic solution proposed in this paper is formulated in Section II, followed by a description of the experimental setup in Section III, and by experimental results in Section IV, presented in Tables I-V.

II. Analytic Solution

To start with, a textbook FNN is vectorized, based on a sigmoid activation function $S(t) = 1/(1+e^{-t})$. The weights $\mathbf{V}$ and $\mathbf{W}$ of such a system (often denoted as W_{IH} versus W_{HO}), may be expressed according to Figs. 1-2. In this representation, defined here as the normal form, the output of the network may

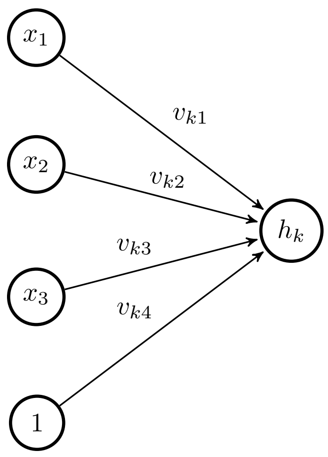

Figure 1. An example with three input nodes ($M = 3$), $h_k = S(\mathbf{v}_k\mathbf{u}) = S(v_{k1}x_1 + v_{k2}x_2 + v_{k3}x_3 + v_{k4})$, using a sigmoid activation function S.

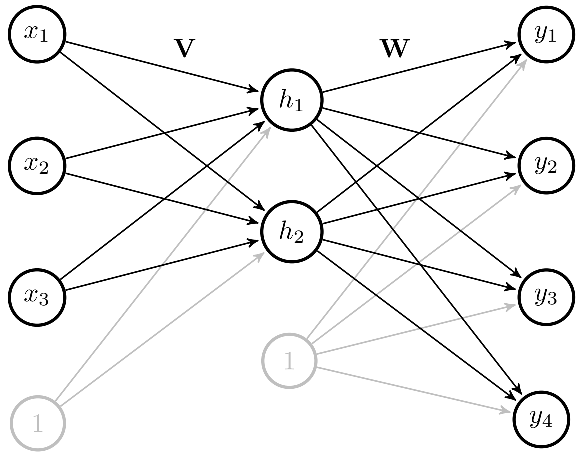

Figure 2. A vectorized model of a standard FNN with a single hidden layer, in this example with $M = 3$ input nodes, $H = 2$ hidden nodes, $K = 4$ output nodes and the weight matrices $\mathbf{V}$ and $\mathbf{W}$, using a sigmoid activation function for the output of each hidden node. In this model, the biases for the hidden layer and the output layer correspond to column $M + 1$ in $\mathbf{V}$ versus column $H + 1$ in $\mathbf{W}$.

be expressed as:

$$\mathbf{y} = \mathbf{W}\mathbf{h} = \mathbf{W}\begin{bmatrix} \dfrac{S(\mathbf{V}\mathbf{u})}{1} \end{bmatrix}, \; \mathbf{u} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{1}$$

where $\mathbf{x} = [x_1 \; x_2 \; \dots \; x_M]^T$ denotes the input signals, $\mathbf{y} = [y_1 \; y_2 \; \dots \; y_K]^T$ the output signals, and S, an element-wise

$$\underbrace{\begin{bmatrix} *&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\left(\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}*&*&1\\ *&*&1\\ *&*&1\\ *&*&1\\ *&*&1\\ *&*&1\end{bmatrix}\right)^{-1}}_{\underbrace{\begin{bmatrix}*&*&*\\ *&*&*\\ *&*&*\end{bmatrix}}_{(\mathbf{UU}^T)^{-1}}} \underbrace{\left[\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}*&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\end{bmatrix}\right]}_{\underbrace{\begin{bmatrix}*&*&*&*&*\\ *&*&*&*&*\\ *&*&*&*&*\end{bmatrix}}_{\mathbf{UH}_0^T}}$$

$$\underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}}_{\mathbf{H}} = \left[\begin{array}{c} S\left(\underbrace{\begin{bmatrix}*&*&*\\ *&*&*\\ *&*&*\\ *&*&*\\ *&*&*\end{bmatrix}}_{\mathbf{V}}\underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}}_{\mathbf{U}}\right) \\ \hline \underbrace{\begin{bmatrix}1&1&1&1&1&1\end{bmatrix}}_{\mathbf{1}^T}\end{array}\right]$$

$$\underbrace{\begin{bmatrix}*&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\end{bmatrix}}_{\mathbf{W}^T} = \underbrace{\left(\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}*&*&*&*&*&1\\ *&*&*&*&*&1\\ *&*&*&*&*&1\\ *&*&*&*&*&1\\ *&*&*&*&*&1\\ *&*&*&*&*&1\end{bmatrix}\right)^{-1}}_{(\mathbf{HH}^T)^{-1}} \underbrace{\left[\underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}}_{\mathbf{H}}\underbrace{\begin{bmatrix}*&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\end{bmatrix}}_{\mathbf{Y}_0^T}\right]}_{\underbrace{\begin{bmatrix}*&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\\ *&*&*&*\end{bmatrix}}_{\mathbf{HY}_0^T}}$$

$$\underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\end{bmatrix}}_{\mathbf{W}}\underbrace{\begin{bmatrix}*&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ *&*&*&*&*&*\\ 1&1&1&1&1&1\end{bmatrix}}_{\mathbf{H}}$$

Figure 3. A visual representation of the evaluation of weights $\mathbf{V}$ and $\mathbf{W}$ by the analytic method presented in this paper, and the actual output $\mathbf{Y}$, in this example as a function of six training points, $N = 6$, the training input and output sets $\mathbf{U}$ and $\mathbf{Y}_0$, with two inputs, $M = 2$, four outputs, $K = 4$, and five hidden nodes, $H = N - 1 = 5$. In this figure, an asterisk denotes a floating-point number. To facilitate bias values, certain matrix elements are set to one.

sigmoid function. In this paper, a winner-take-all classification model is used, where the final output of the network is the selection of the output node that has the highest value. Since the sigmoid function is a constantly increasing function and identical for each output node, it can be omitted from the output layer, as $\max(\mathbf{y})$ results in the same node selection as $\max(S(\mathbf{y}))$. Further on, presuming that the training set is highly fragmented (the input-output relations in the training sets were in our experiments established by a random number generator), denoting N as the number of training points, the number of hidden nodes is preferred to be set to $H = N - 1$. Defining a batch (training set), the input matrix $\mathbf{U}$, may be expressed as:

$$\mathbf{U} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MN} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2}$$

where column vector i in $\mathbf{U}$, corresponds to training point i, column vector i in $\mathbf{Y}_0$ (target output value) and in $\mathbf{Y}$ (actual output value). Further, defining $\mathbf{H}$ of size $N \times N$, as the batch values for the hidden layer, given a training set of input and output values and $M^+ = M + 1$, the following relations hold:

$$\mathbf{U} = \left[\frac{\mathbf{X}}{\mathbf{1}^T}\right] : [M^+ \times N] \tag{3}$$

$$\mathbf{H} = \left[\frac{S(\mathbf{VU})}{\mathbf{1}^T}\right] : [N \times N] \tag{4}$$

$$\mathbf{Y} = \mathbf{WH} : [K \times N] \tag{5}$$

To evaluate the weights of this network analytically, we need to evaluate the target values (points) of $\mathbf{H}_0$ for the hidden layer. In this paper, the initial assumption is that any point is feasible, as long as it is unique for each training set. Therefore, in this model, $\mathbf{H}_0$ is merely composed of random numbers. Thus, the following evaluation scheme is preliminary suggested for the analytic solution of the weights of such network:

$$\mathbf{V}^T = (\mathbf{UU}^T)^{-1}\mathbf{UH}_0^T : [M^+ \times H] \tag{6}$$

$$\mathbf{W}^T = (\mathbf{HH}^T)^{-1}\mathbf{HY}_0^T : [N \times K] \tag{7}$$

where a least square solution is used for the evaluation of each network weight matrix. Such an equation is nominally expressed as $\mathbf{Ax} = \mathbf{b}$, with the least square solution [2]:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{8}$$

where (8) corresponds to input row vectors $\mathbf{u}_i^T$. It is the use of *column* input vectors, $\mathbf{u}_i$, that yield the expressions found in (6)-(7). Since the mathematical expressions for the analytic solution of the weights of a neural network may still be difficult to follow, an attempt has been made in Fig. 3 to visualize the matrix operations involved. Note that while a nonlinear activation function (such as the sigmoid function) is vital for the success of such network, the inclusion of a bias is not essential. It is for instance possible to omit the biases and to replace $\mathbf{H}_0$ with an identity matrix $\mathbf{I}$. Such configuration would instead yield the following formula for the evaluation of $\mathbf{V}$ and $\mathbf{H}$ (where $\mathbf{UI}$ can further be simplified as $\mathbf{U}$):

$$\mathbf{V}^T = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{UI} : [M^+ \times N] \tag{9}$$

$$\mathbf{H} = S(\mathbf{VU}) : [H \times N] \tag{10}$$

III. Experimental Setup

To test the solution presented in this paper, a minimal mathematical engine was developed in C++, with the capability to solve $\mathbf{X}$ in a linear matrix equation system of the form:

$$\mathbf{AX} = \mathbf{B} \tag{11}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{X}$ denote matrices of appropriate sizes, since it is numerically more efficient to solve a linear equation system directly, than by matrix inversion. In this system, the column vectors of $\mathbf{X}$ are evaluated using a single Gauss-Jordan elimination cycle, where each column vector $\mathbf{x}_i$ in $\mathbf{X}$ corresponds to the column vector $\mathbf{b}_i$ in $\mathbf{B}$, thereby increasing the evaluation speed compared with standard equation solvers.

Backpropagation was in our experiments implemented in C++, using the code presented by Jones [4], as a reference. To measure the efficiency of the new method compared with the standard method (backpropagation), we used a typical definition for the mean-squared error:

$$\epsilon = \frac{1}{2N}\sum_i^K\sum_j^N (a_{ij} - y_{ij})^2 \tag{12}$$

where a_{ij} denotes an element in $\mathbf{Y}_0$ (target value), and y_{ij} the corresponding element in $\mathbf{Y}$ (actual value). Although the new method does not intrinsically benefit from such definition (since there is no need here for differentiation of the mean-squared error, which is however useful for backpropagation), to simplify comparison in Tables I-V, the same definition of the mean-squared error was also used for the new method.

IV. Experimental Results

The experimental results presented in this paper, as shown in Tables I-V, are based on ten individual experiments for each parameter setting using different random seeds, where $\bar{t}$ denotes average execution time, using a single CPU-core on a modern laptop computer, $\bar{\epsilon}$, the average value of the mean-squared errors, and $\tilde{\epsilon}$, the median value. The success rate, $\bar{s}$, is similarly based on an average value. Since the variation of the results is large between the experiments, the average values are in general larger than the median values. If the number of experiments per parameter setting is increased, according to our experiments, the average value tends to increase as well.

On a note of preliminary experiments with respect to robustness, regarding the generalization abilities of the network, the new method showed to steeply lose accuracy with the addition of noise to the input values, compared with a network trained by backpropagation. This shows that although the results seem to be in order according to the tables presented in this paper, the new method lacks robustness for direct use. Further experiments showed however that even small measures, such as an increase in the input range of the network by doubling the size of the training set with the addition of perturbation and a more conscious design of $\mathbf{H}_0$, by for instance the clustering of the random values as a function of the output values, or for layers with few hidden nodes, the binary encoding [3] of $\mathbf{H}_0$, led to significant improvements of the robustness of the new method. This is an encouraging sign, since the sizes of $\mathbf{UU}^T$ and $\mathbf{HH}^T$ are as shown in Fig. 3, independent of N (assuming H is kept intact as N is increased). There is thus a chance that these robustness issues can be solved, in this context, without any significant impact to the computational speed of the new method.

V. Conclusion

The method proposed in this paper is fast, accurate for networks with a sufficient number of hidden nodes, and straightforward to implement; but is, at this stage, based on preliminary robustness tests, significantly much less robust (thus, resembling an overtrained network), compared with a well-trained FNN through backpropagation. Even small modifications of the new method showed however to increase robustness significantly, which is promising for further research.

References

[1] R. P. W. Duin, “Learned from Neural Networks”, ASCI2000, Lommel, Belgium, 2000, pp. 9-13.

[2] C. H. Edwards and D. E. Penney, Elementary Linear Algebra, Prentice Hall, 1988.

[3] F. Gray, Pulse Code Communication, Patent, U.S., no. 2632058, 1947.

[4] M. T. Jones, AI Application Programming, 2nd ed., Charles River, 2005.

[5] Matlab, The MathWorks, Inc. <http://www.mathworks. com/> [retrieved: April 11, 2014].

[6] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3nd ed., Prentice Hall, 2009.

[7] Y. Yam, “Accelerated Training Algorithm for Feedforward Neural Networks Based on Least Squares Method”, Neural Processing Letters, vol. 2, no. 4, 1995, pp. 20-25.

Table I. BACKPROPAGATION WITH $H = N - 1$ AND AVERAGE SUCCESS RATE $\bar{s}$

M	N	H	K	Iterations	$\bar{t}$	$\bar{\epsilon}$	$\tilde{\epsilon}$	$\bar{s}$ (%)
5	20	19	5	10^4	46.6 ms	0.0671	0.0620	93.0
5	20	19	5	10^6	4.39 s	0.0175	$4.64 \cdot 10^{-5}$	96.5
10	50	49	10	10^4	182 ms	0.116	0.114	83.2
10	50	49	10	10^6	18.1 s	0.0680	0.0650	86.4
20	50	49	20	10^4	333 ms	0.0394	0.0392	94.6
20	50	49	20	10^6	33.3 s	0.0170	0.0200	96.6
40	100	99	40	10^4	1.27 s	0.0671	0.0670	88.9
40	100	99	40	10^6	127 s	0.0180	0.0200	96.4

Table II. NEW METHOD WITH $H = N - 1$

M	N	H	K	$\bar{t}$	$\bar{\epsilon}$	$\tilde{\epsilon}$	$\bar{s}$ (%)
5	20	19	5	332 μs	$3.34 \cdot 10^{-8}$	$2.00 \cdot 10^{-13}$	100.0
10	50	49	10	3.74 ms	$6.99 \cdot 10^{-9}$	$2.26 \cdot 10^{-12}$	100.0
20	50	49	20	4.79 ms	$2.68 \cdot 10^{-13}$	$5.93 \cdot 10^{-16}$	100.0
40	100	99	40	36.7 ms	$3.93 \cdot 10^{-12}$	$2.33 \cdot 10^{-13}$	100.0

Table III. BACKPROPAGATION WITH $H < N - 1$ AND 10^4 ITERATIONS

M	N	H	K	$\bar{t}$	$\bar{\epsilon}$	$\tilde{\epsilon}$	$\bar{s}$ (%)
5	20	5	5	14.7 ms	0.130	0.125	86.5
10	50	10	10	43.1 ms	0.227	0.237	71.6
20	50	20	20	144 ms	0.0624	0.0599	94.2
40	100	40	40	531 ms	0.115	0.115	84.8

Table IV. NEW METHOD WITH $H < N - 1$

M	N	H	K	$\bar{t}$	$\bar{\epsilon}$	$\tilde{\epsilon}$	$\bar{s}$ (%)
5	20	5	5	59 μs	0.281	0.289	58.5
10	50	10	10	370 μs	0.350	0.350	50.0
20	50	20	20	1.30 ms	0.279	0.277	91.8
40	100	40	40	9.24 ms	0.290	0.290	98.5

Table V. SELECTIVE USE OF BIAS FOR THE NEW METHOD, WITH $M = 40$, $N = 100$, $H = 99$, AND $K = 40$

H_b	Y_b	$\bar{t}$	$\bar{\epsilon}$	$\tilde{\epsilon}$	$\bar{s}$ (%)
No	No	35.7 ms	0.00514	0.00513	100.0
No	Yes	36.4 ms	$1.35 \cdot 10^{-12}$	$1.91 \cdot 10^{-14}$	100.0
Yes	No	36.2 ms	0.00544	0.00534	100.0