Method for Analytic Evaluation of the Weights of a Robust Large-Scale Multilayer Neural Network with Many Hidden Nodes

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

***Abstract*—The multilayer feedforward neural network is presently one of the most popular computational methods in computer science. The current method for the evaluation of its weights is however performed by a relatively slow iterative method known as backpropagation. According to previous research, attempts to evaluate the weights analytically by the linear least square method, showed to accelerate the evaluation process significantly. The evaluated networks showed however to fail in robustness tests compared to well-trained networks by backpropagation, thus resembling overtrained networks. This paper presents the design and verification of a new method, that solves the robustness issues for a large-scale neural network with many hidden nodes, as an upgrade to the previously suggested analytic method.**

Keywords-analytic; FNN; large-scale; least square method; neural network; robust; 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 (FNN), where the weights are estimated by an iterative training method called backpropagation [7][9]. Although this iterative method is relatively fast for small networks, it is rather slow for large ones, given the computational power of modern computers [1][2]. To accelerate the training speed of FNNs, many approaches have been suggested based on the least square method [3]. 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 [8][10].

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 a *robust analytic solution* (i.e., with good generalization abilities compared with a well-trained network using backpropagation, but without any iterations involved), for the weights of an FNN, that is easily understood and that may be implemented relatively effortlessly, using a mathematical application such as Matlab [6].

In a previous work [4], an analytic solution was proposed for the evaluation of the weights of a textbook FNN. This solution was found to be significantly much faster, and for $H = N - 1$ (where H denotes the number of hidden nodes and N, the number of training points), more accurate than solutions provided by backpropagation, but at the same time significantly less robust compared with a well-trained network using backpropagation, why the analytic solution was considered to lack 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

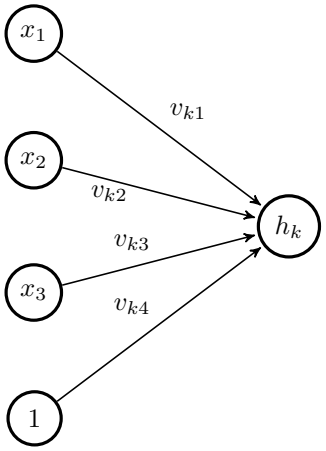

Figure 1. An example with three input nodes ($M = 3$), $h_k = S(\mathbf{v}_k\mathbf{u}) = S(v_{k1}x_1 + v_{k2}x_2 + v_{k3}x_3 + v_{k4})$, using a sigmoid activation function S.

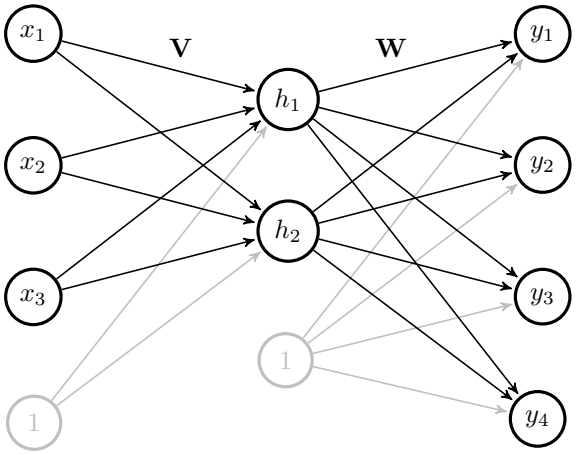

Figure 2. A vectorized model of a standard FNN with a single hidden layer, in this example with $M = 3$ input nodes, $H = 2$ hidden nodes, $K = 4$ output nodes and the weight matrices $\mathbf{V}$ and $\mathbf{W}$, using a sigmoid activation function for the output of each hidden node. In this model,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

 ISBN: 978-1-61208-367-4

perturbation, led to significant improvement of the robustness of the evaluated network. As a first systematic attempt to address the issue of robustness, this paper presents the derivation, implementation and verification of a new method, based on the expansion of the training set of an FNN, with addition of perturbation, but in practice without any impact on the execution speed of the original method. As a brief overview, in Section II, a recap is made of the theory behind the analytic method presented in [4], which is the foundation of the theory presented in this paper. In Section III, a new method (or upgrade) is derived for the improvement of the robustness of the original method. In Section IV, the experimental setup is briefly described, and in Section V, the new method is experimentally verified by comparison with the performance of the original one.

II. Analytic Solution

In [4], 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 system (often denoted as W_{IH} versus W_{HO}), may be represented by Figures 1-2. In this representation, defined here as the normal form, the output of the network may 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 $\mathbf{x} = [x_1\ x_2\ \ldots\ x_M]^T$ denotes the input signals, $\mathbf{y} = [y_1\ y_2\ \ldots\ y_K]^T$ the output signals, and S, an element-wise 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 constantly increasing 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), 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:

$$\mathbf{U} = \begin{bmatrix}\dfrac{\mathbf{X}}{\mathbf{1}^T}\end{bmatrix} : [M^+ \times N] \tag{3}$$

$$\mathbf{H} = \begin{bmatrix}\dfrac{S(\mathbf{V}\mathbf{U})}{\mathbf{1}^T}\end{bmatrix} : [N \times N] \tag{4}$$

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

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 context, 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 suggested for the analytic solution of the weights of such network:

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

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

where a least square solution is used for the evaluation of each network weight matrix. Such equation is nominally expressed as:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{8}$$

with the least square solution [3]:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{9}$$

Since the mathematical expressions for the analytic solution of the weights of a neural network may be difficult to follow, an attempt has been made in Figure 3 to visualize the matrix operations involved. 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 a configuration would instead yield the following formula for the evaluation of $\mathbf{V}$ and $\mathbf{H}$ (where $\mathbf{U}\mathbf{I}$ can further be simplified as $\mathbf{U}$):

$$\mathbf{V}^T = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{U}\mathbf{I} : [M^+ \times N] \tag{10}$$

$$\mathbf{H} = S(\mathbf{V}\mathbf{U}) : [H \times N] \tag{11}$$

III. Proposal

To start with, we expand the input training set $\mathbf{U}$ in (2), by the addition of perturbation to the input signal, given the definition $\mathbf{\Theta} = \mathbf{U}\mathbf{U}^T$, with $\theta_{M^+M^+} = N$ in:

$$\mathbf{\Theta} = \begin{bmatrix} \theta_{11} & \theta_{12} & \ldots & \theta_{1M} & \theta_{1M^+} \\ \theta_{21} & \theta_{22} & \ldots & \theta_{2M} & \theta_{2M^+} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \theta_{M1} & \theta_{M2} & \ldots & \theta_{MM} & \theta_{MM^+} \\ \theta_{M^+1} & \theta_{M^+2} & \ldots & \theta_{M^+M} & N \end{bmatrix} \tag{12}$$

Further, an extended matrix $\tilde{\mathbf{U}}$ is introduced, where:

$$\tilde{\mathbf{U}} = \begin{bmatrix}\ \tilde{\mathbf{U}}_1 & \tilde{\mathbf{U}}_2 & \ldots & \tilde{\mathbf{U}}_N\ \end{bmatrix} \tag{13}$$

with:

$$\mathbf{U}_j = \begin{bmatrix} u_{1j}+\Delta & u_{1j}-\Delta & u_{1j} & u_{1j} \\ u_{2j} & u_{2j} & u_{2j}+\Delta & u_{2j}-\Delta \\ \vdots & \vdots & \vdots & \vdots \\ u_{Mj} & u_{Mj} & u_{Mj} & u_{Mj} \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} \cdots & u_{1j} & u_{1j} \\ \cdots & u_{2j} & u_{2j} \\ \ddots & \vdots & \vdots \\ \cdots & u_{Mj}+\Delta & u_{Mj}-\Delta \\ \cdots & 1 & 1 \end{matrix} \tag{14}$$

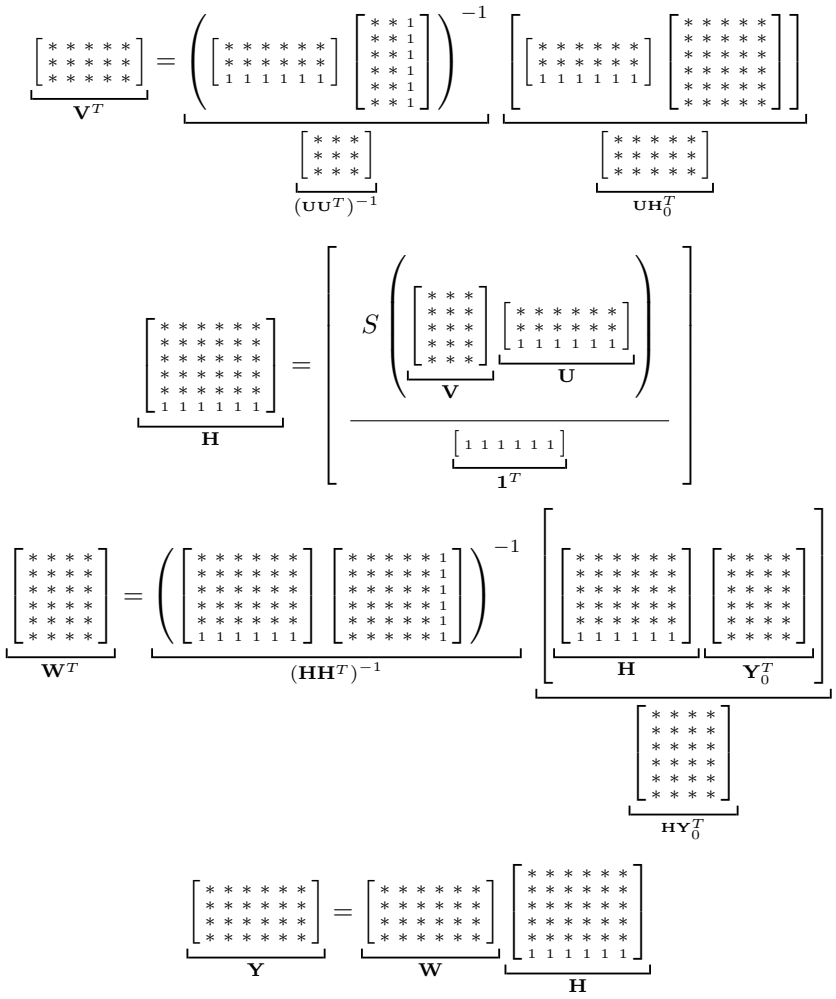

Figure 3. A visual representation of the evaluation of weights $\mathbf{V}$ and $\mathbf{W}$ by the analytic method presented in the related work [4], and the actual output $\mathbf{Y}$, in this example as a function of six training points, $N = 6$, the training input and output sets $\mathbf{U}$ and $\mathbf{Y}_0$, 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.

and where Δ is defined as the amplitude of the perturbation. Thus, for the right hand side of (8), $\tilde{\mathbf{\Theta}} = \tilde{\mathbf{U}}\tilde{\mathbf{U}}^T$, or more explicitly:

$$\tilde{\mathbf{\Theta}} = 2M \begin{bmatrix} d_1 & \theta_{12} & \dots & \theta_{1M} & \theta_{1M^+} \\ \theta_{21} & d_2 & \dots & \theta_{2M} & \theta_{2M^+} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \theta_{M1} & \theta_{M2} & \dots & d_M & \theta_{MM^+} \\ \theta_{M^+1} & \theta_{M^+2} & \dots & \theta_{M^+M} & N \end{bmatrix} \tag{15}$$

with $d_i = \theta_{ii} + \alpha$, where $\alpha = N\Delta^2/M$, or:

$$\tilde{\mathbf{\Theta}} = 2M \left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \dots, \alpha, 0)\right] \tag{16}$$

where $\text{diag}(d_1, d_2, \dots, d_{M^+})$, denotes a diagonal matrix of size $M^+ \times M^+$ (where $M^+ = M + 1$), with the diagonal elements $d_1, d_2, \dots, d_{M^+}$. Similarly, for the left hand side of (8), $\mathbf{\Psi}$ and $\mathbf{\Lambda}$ are defined as:

$$\mathbf{H}_0^T = \mathbf{\Psi} = \begin{bmatrix} \psi_{11} & \psi_{12} & \dots & \psi_{1H} \\ \psi_{21} & \psi_{22} & \dots & \psi_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{N1} & \psi_{N2} & \dots & \psi_{NH} \end{bmatrix} \tag{17}$$

$$\mathbf{\Lambda} = \mathbf{U}\mathbf{H}_0^T = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \dots & \lambda_{1H} \\ \lambda_{21} & \lambda_{22} & \dots & \lambda_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{M+1} & \lambda_{M+2} & \dots & \lambda_{M+H} \end{bmatrix} \tag{18}$$

and thereby, $\mathbf{\Psi}$ and $\mathbf{\Psi}_j$ as:

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{\Psi}_1 \\ \mathbf{\Psi}_2 \\ \vdots \\ \mathbf{\Psi}_N \end{bmatrix} \tag{19}$$

$$\mathbf{\Psi}_j = \begin{bmatrix} \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \\ \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \end{bmatrix} \tag{20}$$

with $\tilde{\mathbf{\Lambda}} = \tilde{\mathbf{U}}\mathbf{\Psi}$:

$$\tilde{\mathbf{\Lambda}} = 2M\mathbf{\Lambda} \tag{21}$$

This transforms (8) into:

$$\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T\mathbf{X} = \tilde{\mathbf{U}}\mathbf{\Psi} \tag{22}$$

 ISBN: 978-1-61208-367-4

or:

$$\tilde{\mathbf{\Theta}}\mathbf{X} = \tilde{\mathbf{\Lambda}} \tag{23}$$

Thus:

$$2M\left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = 2M\mathbf{\Lambda} \tag{24}$$

Given the matrix equation:

$$\mathbf{AX} = \mathbf{B} \tag{25}$$

since, given a scalar $c \in \mathbb{R}$:

$$c \cdot (\mathbf{AX}) = (c \cdot \mathbf{A})\mathbf{X} = c \cdot \mathbf{B} \tag{26}$$

thereby:

$$\left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = \mathbf{\Lambda} \tag{27}$$

This yields thus, the final expression:

$$\left[\mathbf{U}\mathbf{U}^T + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = \mathbf{U}\mathbf{H}_0^T \tag{28}$$

Hence, the expansion of $\mathbf{U}$ into a perturbation matrix $\tilde{\mathbf{U}}$ of size $M^+ \times 2MN$, and similarly of $\mathbf{H}_0^T$ into a matrix $\mathbf{\Psi}$ of size $2MN \times H$, is according to (28), equivalent to the reinforcement of the diagonal elements of the square matrix $\mathbf{\Theta} = \mathbf{U}\mathbf{U}^T$, by the addition of a factor $\alpha = N\Delta^2/M$ to each diagonal element, except for the last one, which as a consequence of the use of bias in the network, is left intact.

IV. Experimental Setup

The experiments presented in this paper are based on a minimal mathematical engine that was developed in C++, with the capability to solve $\mathbf{X}$ in a linear matrix equation system of the form $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{X}$ denote matrices of appropriate sizes, since it is computationally 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 [3], where each column vector $\mathbf{x}_i$ in $\mathbf{X}$ corresponds to the column vector $\mathbf{b}_i$ in $\mathbf{B}$. Backpropagation was in these experiments, for high execution speed (and a fair comparison with the new methods), also implemented in C++, using the code presented in [5] as a reference.

V. Results

The experimental results presented in this paper are shown in Figures 4-9, measuring average success rate, and Table I, measuring execution speed. Each experiment is based on ten individual experiments (with different random seeds), using a single CPU-core on a modern laptop computer. In Table I, $\bar{t}_{\text{bp}}$ denotes the execution time for backpropagation based on 10000 iterations, which applies to all backpropagation experiments presented in this paper. Similarly, $\bar{t}_{\text{new}}$ denotes the execution time for the original analytic method in [4], and $\bar{t}_{\text{new+}}$, the execution time for the new method presented in this paper, using diagonal reinforcement.

In these experiments, the input values to the FNN consisted of the integers $\{0, 1, 2\}$, and the output values of a binary number, $\{0, 1\}$. To avoid inconsistencies (or repetition) in any training set, identical input values were replaced by unique values. For the addition of noise, a random value (with uniform distribution) in the range of $\pm\Delta$, was added to each input value. However, although according to our derivation of (28), $\alpha = N\Delta^2/M$, α had in practice to be retuned to $10^5 \cdot N\Delta^2$ for good results in the experiments in Figures 4-6 ($H = N - 1$), and to $10^4 \cdot N\Delta^2$ in Figures 7-9 (few hidden nodes).

TABLE I. Average execution time

Figure	M	N	H	K	$\bar{t}_{\text{bp}}$	$\bar{t}_{\text{new}}$	$\bar{t}_{\text{new+}}$
4	10	25	24	10	92.5 ms	688 μs	668 μs
5	20	50	49	20	332 ms	4.84 ms	4.86 ms
6	40	100	99	40	1.27 s	37.0 ms	37.1 ms
7	10	25	10	10	43.3 ms	212 μs	202 μs
8	20	50	20	20	144 ms	1.33 ms	1.31 ms
9	40	100	40	40	535 ms	9.35 ms	9.38 ms

VI. Conclusion

The upgrade proposed in this paper, showed to solve the robustness issues of the analytic solution of the weights of a large-scale FNN with $H = N - 1$ nodes, and in practice without any impact on the execution speed of the solution. Since according to [4], the original method was not considered to be ready for direct use until the robustness issues had been solved, this upgrade provides hereby a method that, given access to a linear equation solver, while considerably faster, is for large-scale networks with many hidden nodes, comparable in robustness to a well-trained FNN by backpropagation.

References

[1] P. De Wilde, Neural Networks Models: An Analysis, Springer, 1996, pp. 35-51.

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

[3] C. H. Edwards and D. E. Penney, Elementary Linear Algebra, Prentice Hall, 1988, pp. 220-227.

[4] M. Fridenfalk, "The Development and Analysis of Analytic Method as Alternative for Backpropagation in Large-Scale Multilayer Neural Networks", The Proceedings of ADVCOMP 2014, Rome, Italy, August 2014, in press.

[5] M. T. Jones, AI Application Programming, 2nd ed., Charles River, 2005, pp. 165-204.

[6] Matlab, The MathWorks, Inc. <http://www.mathworks.com/> [retrieved: August 23, 2014].

[7] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3nd ed., Prentice Hall, 2009, pp. 727-736.

[8] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation", The Proceedings of IEEE, vol. 78, no. 9, 1990, pp. 1415-1442.

[9] B. J. Wythoff, "Backpropagation Neural Networks: A Tutorial", Chemometrics and Intelligent Laboratory Systems, vol. 18, no. 2, 1993, pp. 115-155.

[10] 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.

 ISBN: 978-1-61208-367-4

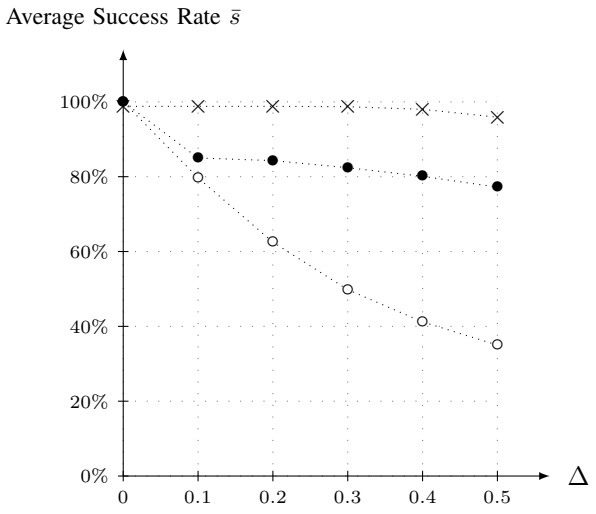

Figure 4. Backpropagation (×), analytic method (○), versus new method using diagonal reinforcement (●), with $M = 10$ (input nodes), $N = 25$ (training points), $H = 24$ (hidden nodes), and $K = 10$ (output nodes).

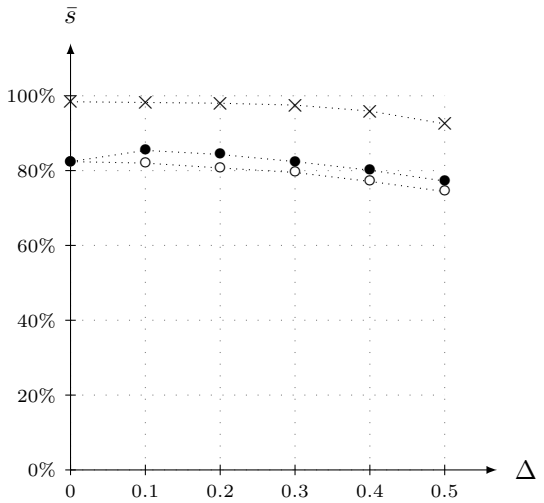

Figure 7. $M = 10$, $N = 25$, $H = 10$, and $K = 10$.

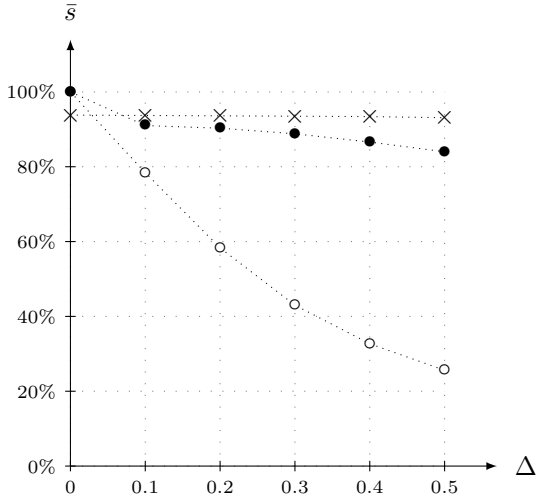

Figure 5. $M = 20$, $N = 50$, $H = 49$, and $K = 20$.

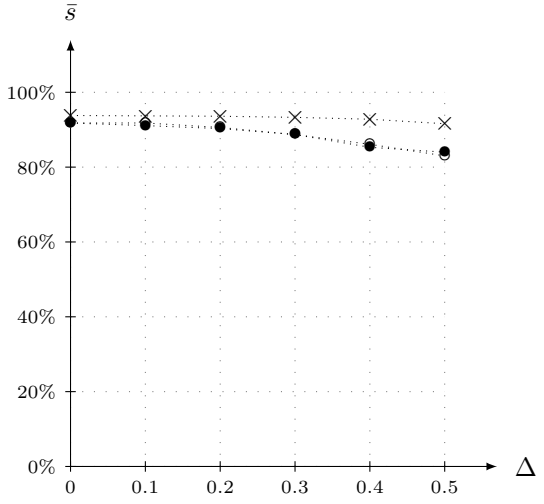

Figure 8. $M = 20$, $N = 50$, $H = 20$, and $K = 20$.

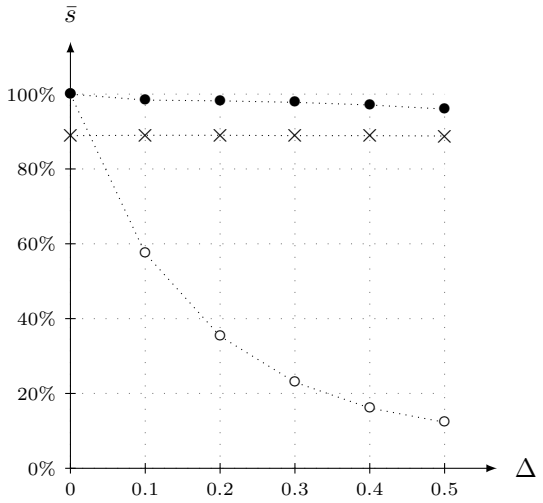

Figure 6. $M = 40$, $N = 100$, $H = 99$, and $K = 40$.

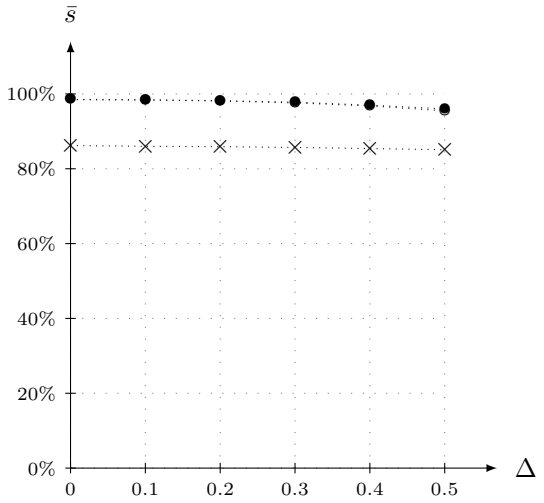

Figure 9. $M = 40$, $N = 100$, $H = 40$, and $K = 40$.

 ISBN: 978-1-61208-367-4