Parameterized Normalization: Application to Wavelet Networks

Richard BARON, Bernard GIRAU

Abstract— **We present a new method to handle the standard data normalization process, which is often used in neural network applications. Normalization is studied in the case of wavelet networks, and we derive a dynamic interpretation of its influence, which can be extended to several neural models. We show that data normalization may be simulated and parameterized to avoid data preprocessing, so that the normalization process becomes either tunable or dynamically adaptable. Both cases are illustrated with benchmark applications. The main benefit of the proposed methods is a big reduction of the time of convergence on a satisfactory classification rate.**

I. INTRODUCTION

Solving concrete problems with the help of neural networks often requires data preprocessing. Input normalization is used in many cases (see [4]). It is usually considered as a method to avoid data scattering so that input patterns better fit the application domain of the transfer functions. In this article, we argue that this static interpretation is not sufficient, since this process may be performed by a simple transformation of the network weights. In support of our claim, we chose to study the wavelet network model, which performance is greatly influenced by normalization. We show that data normalization balances the respective update speeds of the different network weights. We introduce a single parameterization of the *normalization rate*. Its interest is first illustrated by the possibility to tune the normalization process without any data modification. It is applied to the Breiman's waves problem and a sonar signal classification problem. Performance increases are obtained, and the convergence times are greatly reduced. Finally, we discuss the interest of a dynamic adaptation of the normalization rate during learning. We propose a heuristics to tune the normalization interval automatically.

II. RELATED WORKS

Though widely used, data normalization is seldom described in experiment reports. Data are often taken "already" normalized ([5]), so that the authors do not even mention it. To our knowledge, no article addresses the problem of choosing the normalization range, nor the possibility to dynamically adapt this range while learning. Nevertheless, the problem of choosing the range of the random initial weights is briefly mentioned in [7]. We will show that this is directly linked with the normalization process, and therefore with the learning rate values, which is not pointed out in [7]. A different problem (the choice of the activation function gain) is addressed in [18], but the approach is similar to ours: it shows this gain may be tuned thanks to the initial weights and the learning rate (though the interest of this tuning method is not clearly established, especially in terms of computation cost).

The influence of the initial weights on the learning process in multiclass problems has been discussed in [15], [2]. It is shown in section V that the dynamic influence of normalization counterbalances a too fast weight decrease mentioned in these papers.

Moreover, several algorithms have been proposed that handle adaptive learning rates ([1], [7], [8], [12], [13], [14], [16], [19]), or that use different rates for different weights ([12], [17]). None of these algorithms makes use of a relation between learning rates and ranges of data and weights. Some of them are simple heuristics: e.g. in [17] where the learning rate adapts to the neurons fan-in. Some others take into account the previous values of the gradient ([8], [7], [12]). Some better principled ones are based on Newton's method and estimate the second order derivative of the error ([13], [16], [19] and [7] partly), or approximate the Lipschitz constant ([14]).

Our work does not aim at providing a similar and better algorithm: it acts "upstream" from the above gradient-based techniques. Tuning the normalization *modifies the shape of the error function itself*, and it results in linear transforms applied to the gradient vectors. Therefore, a simple gradient descent algorithm as well as most of these advanced algorithms may take advantage of our approach of the normalization process *in addition to* their own optimization method, by taking into account the linearly transformed gradient (though further work is needed to see how normalization exactly affects the hessian for the methods that use it explicitly). For sake of simplicity, this paper only deals with parameterized normalization applied to simple gradient descent. Additional experiments are required to analyse the influence of a tunable normalization on the performance of the above algorithms.

III. WAVELET NEURAL NETWORKS

Recently, *wavelet theory* has been related to neural networks in [20]. A scalar wavelet function ψ is a real function which is localized in space and frequency and which Fourier transform $\hat{\psi}$ satisfies the condition $\int_0^{+\infty} \frac{|\hat{\psi}(\nu)|^2}{|\nu|} d\nu < \infty$.

In the multidimensional case a wavelet function

R. Baron and B. Girau are with the Laboratoire de l'Informatique du parallélisme, ENSL, 46 allée d'Italie, 69364 Lyon cedex 7, France. E-mail: rbaron@ens-lyon.fr, bgirau@ens-lyon.fr

R. Baron is with the Departamento de Ingenieria Matematica, Universidad de Chile, Casilla 170/3 - Correo 3 Santiago Chile

B. Girau is with the Centre Suisse d'Electronique et de Microtechnique, Jaquet-Droz 1, CH-2007 Neuchâtel, Switzerland

0-7803-4859-1/98 $10.00©1998 IEEE

$\Psi : \mathbb{R}^n \mapsto \mathbb{R}$ may be built as a *tensor product* by setting

$$\Psi(\vec{x}) = \prod_{j=1..n} \psi(x_j) \quad \text{for} \quad \vec{x} = (x_1, \ldots, x_n)$$

where ψ is a scalar wavelet. The theory of *discrete wavelet decomposition* (see for example [6]) shows that any function $F \in \mathcal{L}^1(\mathbb{R}^n, \mathbb{R})$ can be approximated with an arbitrary precision as follows:

$$F(\vec{x}) \approx \sum_{i=1}^{N} w^{(i)} \cdot \Psi(D^{(i)}(\vec{x} - \vec{t}^{(i)})) + \theta \qquad (1)$$

where

- $D^{(i)}$: dilation diagonal matrix with diagonal coefficients $d_j^{(i)}$,
- $\vec{t}^{(i)} = \left(t_j^{(i)}\right)_{j=1..n}$: translation vector.

This expression may be written as the output of a *wavelet neural network* with one hidden layer that contains N cells called wavelet neurons. For classification purposes, the output layer may also apply a sigmoid function σ to the weighted sum of the wavelet neurons outputs.

The wavelet network model is a learnable neural network:

- The learnable parameters in the output layer are the weights $w^{(i)}$ and the threshold θ.
- Neuron i in the hidden layer computes

$$\Psi(D^{(i)}(\mathbf{x} - \mathbf{t^{(i)}})) = \prod_{i=1..n} \psi(d_j^{(i)}(x_j - t_j^{(i)}))$$

  Its learnable parameters are the dilation factors $d_j^{(i)}$ and the translation factors $t_j^{(i)}$.

As for any feedforward differentiable neural network (see the general study of differentiability and gradient backpropagation in [9], [10]), the parameters of a wavelet network can be learned with any gradient based learning algorithm (for example gradient descent). We just mention here the differentials w.r.t. the wavelet neuron parameters. Let K be the number of learning patterns. Given a pattern $(\vec{x}^{(k)}, F(\vec{x}^{(k)}))$ $(1 \leq k \leq K)$, we use $c(\vec{x}^{(k)}) = \frac{1}{2}(\sigma(y^{(k)}) - F(\vec{x}^{(k)}))^2$ as cost function, where $\sigma(y^{(k)})$ is the output of the wavelet network with input $\vec{x}^{(k)}$. Then

$$\frac{\partial c}{\partial d_j^{(i)}}(\vec{x}^{(k)}) = (x_j^{(k)} - t_j^{(i)}) T_{i,j}^{(k)} \qquad (2)$$

$$\frac{\partial c}{\partial t_j^{(i)}}(\vec{x}^{(k)}) = -d_j^{(i)} T_{i,j}^{(k)} \qquad (3)$$

where

$$T_{i,j}^{(k)} = (\sigma(y^{(k)}) - F(\vec{x}^{(k)}))\sigma'(y^{(k)}) w^{(i)} \partial_j \Psi(D^{(i)}(\vec{x}^{(k)} - \vec{t}^{(i)}))$$

IV. Simulated normalization

This section presents a generalized version of the preliminary work performed in chapter 6 of [3].

A. Weight initialization

The standard data base normalization preprocessing can be directly simulated by a correct dilation and translation parameter setting.

We consider for example a normalization between a and b for each coordinate axis. If $M_j = \max_{k=1..K} x_j^{(k)}$ and $m_j = \min_{k=1..K} x_j^{(k)}$, then the normalized data are

$$X_j^{(k)} = \frac{b-a}{M_j - m_j} x_j^{(k)} + \frac{aM_j - bm_j}{M_j - m_j} \qquad (4)$$

Therefore, if some parameters $(D^{(i)}, \vec{t}^{(i)})$ are defined w.r.t. other parameters $(D^{(i)\prime}, \vec{t}^{(i)\prime})$, by means of

$$d_j^{(i)} = \frac{b-a}{M_j - m_j} d_j^{(i)\prime} \qquad (5)$$

and

$$t_j^{(i)} = \frac{M_j - m_j}{b-a} t_j^{(i)\prime} - \frac{aM_j - bm_j}{b-a} \qquad (6)$$

then

$$\sigma\left(\sum_{i=1}^{N} w^{(i)} \Psi(D^{(i)\prime}.(\mathbf{X}^{(k)} - \mathbf{t}^{(i)\prime})) + \theta\right) \qquad (7)$$

$$= \sigma\left(\sum_{i=1}^{N} w^{(i)} \Psi(D^{(i)}.(\mathbf{x}^{(k)} - \mathbf{t}^{(i)})) + \theta\right)$$

Interpretation: a wavelet network that uses $d_j^{(i)\prime}$ and $t_j^{(i)\prime}$ as dilation and translation parameters with normalized data, will output the same values as a wavelet network that uses $d_j^{(i)}$ and $t_j^{(i)}$ with unchanged data.

But formula (5,6) gives a static equivalence: when a gradient descent learning is applied, the equivalence between $(D^{(i)\prime}, \vec{t}^{(i)\prime})$ and $(D^{(i)}, \vec{t}^{(i)})$ is not maintained.

B. Learning rates

The equivalence (7) may become dynamic if the learning rates are also adapted. Let ϵ be the learning rate (used for the output layer parameters).

If the learning rates for $d_j^{(i)}$ and $t_j^{(i)}$ are respectively

$$\epsilon_{d_j} = \epsilon \frac{(b-a)^2}{(M_j - m_j)^2} \qquad \epsilon_{t_j} = \epsilon \frac{(M_j - m_j)^2}{(b-a)^2} \qquad (8)$$

then after any number of learning steps, formula (7) will be still satisfied.

SKETCH OF PROOF: Assume formula (7) is satisfied. If a gradient learning step is applied to $(D^{(i)\prime}, \vec{t}^{(i)\prime})$ with ϵ as learning rate and $\vec{X}^{(k)}$ as input, then the next output of neuron i in the hidden layer is (with simplified notations):

$$\prod_{i=1..n} \psi((d_j^{(i)\prime} - \epsilon \frac{\partial c}{\partial d_j^{(i)\prime}})(X_j - (t_j^{(i)\prime} - \epsilon \frac{\partial c}{\partial t_j^{(i)\prime}}))) \qquad (9)$$

Equations (2,3) and (5,6) lead to

$$\frac{\partial c}{\partial d_j^{(i)\prime}}(\vec{X}^{(k)}) = \frac{b-a}{M_j - m_j}\frac{\partial c}{\partial d_j^{(i)}}(\vec{x}^{(k)})$$

$$\frac{\partial c}{\partial t_j^{(i)\prime}}(\vec{X}^{(k)}) = \frac{M_j - m_j}{b-a}\frac{\partial c}{\partial t_j^{(i)}}(\vec{x}^{(k)})$$

So that introducing formulas (4,5,6,8) in (9) leads to

$$\prod_{i=1..n} \psi((d_j^{(i)} - \epsilon_{d_j}\frac{\partial c}{\partial d_j^{(i)}})(x_j - (t_j^{(i)} - \epsilon_{t_j}\frac{\partial c}{\partial t_j^{(i)}})))$$

which is the output of neuron i using $(D^{(i)}, \vec{t}^{(i)})$ after one learning step with modified learning rates (8), when the input is $\vec{x}^{(k)}$.

The use of formula (5,6) (for initialization) and (8) is a *simulated* normalization. It does not require any data modification.

V. Dynamic interpretation of normalization

The normalization process is usually seen as a way to obtain "less spread" data. It is considered as a static preprocessing that makes the data *fit* the activity domain of the neurons. This static point of view does not justify the normalization interest, since this "fitting" process can be inversely performed by making the neurons fit the data (equations (5,6)).

The main influence of a normalization in a neural network application is indeed dynamic. Due to localization properties, significant values of $t_j^{(i)}$ approximately range from m_j to M_j. The scalar wavelet function in use can be considered negligible when too much or conversely not enough dilated. Then, significant values of $d_j^{(i)}$ range over some domain $[\frac{k_1}{M_j - m_j}, \frac{k_2}{M_j - m_j}]$.

We define β_j as the ratio of both translation and dilation ranges, that is $\beta_j = \frac{(M_j - m_j)^2}{k_2 - k_1}$. Assume a standard gradient descent is used. Then equation (2) implies that the update term magnitude of $d_j^{(i)}$ w.r.t. its own range is proportional to β_j. In the same way equation (3) shows that the update term magnitude of $t_j^{(i)}$ w.r.t. its own range is proportional to $\frac{1}{\beta_j}$.

Thus, data spreading ($M_j - m_j \gg 1$) makes the learning of the translation coefficients "slow down" and speeds up the learning of the dilation coefficients (whereas it is neutral for the other parameters).

As shown by equation (8), normalization counterbalances the effect of data spreading by reducing the dilation learning rate and increasing the translation learning rate.

Our approach is not limited to the case presented in section IV. For instance, we may handle other kinds of normalization, e.g. a simple vector homothety. Moreover, equations (5,6) and (8) can be derived for other neural network models. For example, in the case of a MLP (multilayer perceptron), neuron i in the first layer computes $\sigma(\sum_{j=1..n} w_j^{(i)} x_j + \theta^{(i)})$. It leads to $\epsilon_{w_j} = \epsilon\frac{(b-a)^2}{(M_j - m_j)^2}$ and $\epsilon_\theta = \epsilon$, so that the normalization process makes the learning of the first layer weights slow down w.r.t. the learning of all other network parameters: normalization helps avoiding the too fast decrease of the weights in the hidden layer, that is mentioned in [15], [2].

Some other standard models require restricted conditions to be similarly handled. For example, only normalization by means of vector homothety may be simulated by the parameters of a standard RBF (radial basis function) network, which neurons compute $g(\frac{\|\vec{x} - \vec{t}^{(i)}\|}{\sigma^{(i)}})$. In this case normalization speeds up the learning of the RBF centers.

VI. Parameterized normalization

General formulas for normalization interval $[a, b]$ are given in (4) and (5,6). Nethertheless most scalar wavelet functions in use are symetrical around 0. We thus wish to normalize data in an origin-centered interval, that is $b = -a$.

If we introduce the parameter $\delta = \frac{1}{4b^2}$, then

$$t_j^{(i)} = (M_j - m_j)\sqrt{\delta}t_j^{(i)\prime} + \frac{M_j + m_j}{2}$$

$$d_j^{(i)} = \frac{d_j^{(i)\prime}}{\sqrt{\delta}(M_j - m_j)}$$

$$\epsilon_{d_j} = \frac{\epsilon}{\delta(M_j - m_j)^2} \qquad \epsilon_{t_j} = \epsilon(M_j - m_j)^2\delta \qquad (10)$$

Notice that δ is a notation related to the normalization interval width. *It will now be used as a real parameterization of the normalization rate.* In the next sections, we first show that this notation allows an easy and useful tuning of the normalization interval, to make it adequate to the handled problem at little cost. We then exhibit preliminary experiments to show the interest of some dynamic tuning of this parameter during the learning phase.

A. Static tuning

The first application of the above discussion is the use of *simulated normalization* instead of the standard preprocessing.

It allows therefore to consider now that normalization depends on an easily *tunable* parameter (δ). Indeed, the choice of the normalization interval is often arbitrary, though it has got a strong influence on both learning time and final performance. Equations (5,6) and (10) allow to tune this "normalization rate" by means of only $2n(N+1)$ computations, whatever the data set size is, whereas a standard normalization requires $2nK$ computations (equation (4)).

Next subsections illustrate the interest of such tuning in two applications.

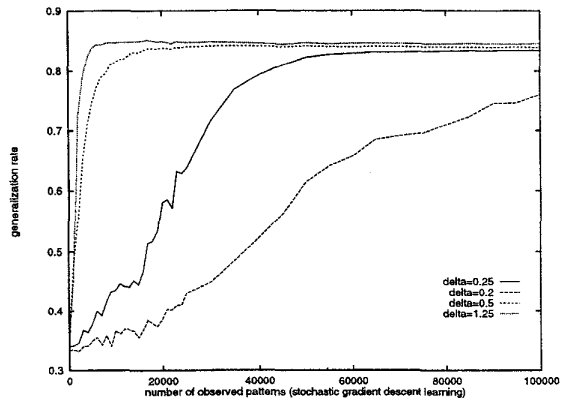

Fig. 1. Static tuning of the normalization rate (Breiman)

A.1 Application to Breiman's waveforms

To illustrate this, we consider now the Breiman's waves problem ([5]), for which wavelet networks have proved efficient ([3]). This problem deals with the classification of three kinds of waveforms. Signals to be classified are 21-dimensional real vectors. The learning set is the one used by Breiman, composed of 300 patterns. The generalization set contains 5000 patterns. According to [5], a classification criterion based on Bayes rules leads to an optimal classification rate equal to 86%.

This problem has been handled in [3] with the following experimental conditions:

- Classification performed by a wavelet network with 5 wavelet neurons (tensor product based on the wavelet $-x\mathrm{e}^{-\frac{x^2}{2}}$).
- On-line (stochastic) gradient descent learning algorithm (100000 iterations) with a learning rate ϵ decreasing from 0.001 to 0.002.
- Data normalization in the interval $[-1;1]$, that is $\delta = 0.25$ in our simulated normalization.

These experimental conditions are our *reference case*.

We have performed several experiments in the same conditions, except for the normalization. Results for the reference case and for $\delta = 0.2$, $\delta = 0.5$ and $\delta = 1.25$ are shown in figure 1. In the reference case, the results are averaged over 10 different random initializations. The results for the other values of δ use the same initializations, to which transformations (5,6) are applied.

In addition to a significant performance increase, the convergence time is greatly reduced when $\delta > 0.4$. The optimal value is $\delta = 1.25$, i.e. a normalization between $-\frac{1}{\sqrt{5}}$ and $\frac{1}{\sqrt{5}}$: after only 5000 learning steps, the classification rate is 83.75%, whereas

- After only 5000 iterations, the classification rate of the reference case is 37.85%.
- 65000 iterations are required in the reference case to reach a similar performance (83%).

A.2 Application to high dimensional data

Moreover, in the case of wavelet networks, high dimensional data raise specific problems ([3]). First, as any local model, wavelet network is subject to the so called *"curse of dimensionality"*: the number of hidden units required to address a given problem grows exponentially with its input space dimension. Second, in the case of the tensor product construction, a specific numerical problem may appear. The value of a scalar wavelet function decrease rapidly. Thus, the value of the function applied to a coordinate, outside some "reasonable" interval, may be very small. This phenomenon is worsened when the value of a multidimensional wavelet function is obtained by a tensor product of these scalar wavelets. In such a case, the search for a correct normalization interval length becomes critical.

This is the case of the sonar signals classification of [11] : signals are to be classified in two classes. They are represented by 60-dimensional real vectors. Both learning and generalization sets are composed of 104 examples. The influence of the value of the normalization rate is illustrated in the following table. Experimental conditions in the reference case ([3]) are similar to the previous case:

- Same wavelet network, same learning algorithm.
- Constant learning rate equal to 0.025.
- $\vec{t}^{(i)}$ randomly initialized among input patterns.

The usual normalization interval $[-1,1]$ ($\delta = 0.25$) leads to bad results (see the following table), so that the choice of a more suitable normalization interval was essential in [3]. This choice might be easily performed by the tuning of δ.

δ	generalization rate
0.25	56.6%
1.25	83.4%
2.5	84.5%
3.75	85.0%
5	83.7%

B. Dynamic tuning

In the previous subsection, the tuning of δ is made before the learning phase. Yet the low cost of simulated normalization even allows to change δ during learning, i.e. to change the normalization interval during learning. Such a *dynamic tuning* may have two main applications:

1. Normalization now adapts to real-time learning, where m_j and M_j may be regularly updated according to patterns newly added to the learning base.

2. Despite its obvious interest, static tuning of the normalization rate requires costly additional experiments. It appears as another learning parameter, to be optimized on an experimental trial-error based method. Thanks to dynamic updates of δ, this tuning might be performed more automatically. We illustrate this with Breiman's waves.

We propose the following heuristics to automatically adapt the normalization rate:

1. Initialization: data are taken unchanged as in the reference case (equivalent to $\delta = 0.25$, since the interval is $[-1;1]$).
2. After each epoch (the whole learning set has been presented), the classification rate r of the learning set is computed.

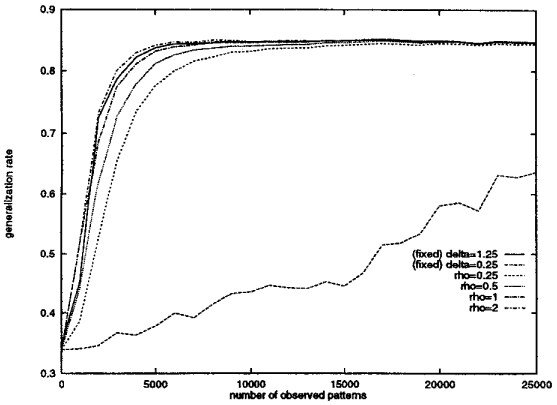

Fig. 2. Dynamic tuning of the normalization rate (Breiman)

If r is lower than the Bayesian expected rate 0.86, then δ is updated as follows:

$$\delta = \delta + \rho(0.86 - r)^2$$

where ρ is a constant value. This corresponds to an increase in the value of δ. According to formulas (10), the translation learning rates increase, and the dilation's ones decrease.

This heuristics for automatic and dynamic tuning of δ is illustrated in figure 2 (since the behaviours are similar, only 25000 iterations are considered).

The main interest is that the observed behaviour does not significantly depend on ρ.

Thus ρ appears as a new parameter, but which value does not affect significantly the final performances. Indeed, the generalization rate evolution with all values of ρ is comparable with the best one obtained for a fixed δ. Therefore no precise prior tuning is then required. The value elected for ρ is far less relevant, thus diminishing the number of sensitive parameters.

As mentioned above, increasing the value of δ causes a speed-up in the translation parameters corrections, relatively to the dilation parameters' ones. This may be compared with the multi-stage training procedure used for RBF networks [4]. The first step usually consists in initializing the centers of the hidden cells. Some *clustering* algorithm is used, so that the model fits the data geometry. The increase of δ might imitate this center initialization by allowing a faster determination of the translation parameters. In the case of the proposed heuristics, this tendancy is maintained as long as the performance is unsatisfactory. The links of this heuristics with RBF center initialization algorithms are now to be examined more closely. This heuristics is empirical. Deriving general ones is now under investigation.

VII. Conclusion

In this article, we have shown that data normalization should not be considered only as a static preprocessing: it allows to balance the correction speeds of the different parameters. We have defined a real parameterization of normalization, based on an equivalent simulated normalization. This analysis is valid for several neural models. It has been illustrated in this paper by experiments using wavelet networks: the normalization tuning has appeared useful, especially to reduce the convergence time. Moreover, the normalization rate may be now easily tuned during learning. It has allowed us to develop a heuristics that automatically tunes the normalization rate according to the error evolution. Such automatic adaptation is still under investigation. We now study the inclusion of the normalization rate in the network, to handle it as a *learnable* shared parameter.

References

[1] S. Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, June 1993.

[2] R. Anand, K. Mehrotr, C.K. Mohnan, and S. Ranka. Efficient classification for multiclass problems using modular neural networks. *IEEE Trans. on Neural Networks*, 6(1):117–124, 1995.

[3] R. Baron. *Contribution à l'étude des réseaux d'ondelettes*. PhD thesis, Ecole Normale Supérieure de Lyon, 1997.

[4] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[5] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees*. 0-534-98054-6. Wadsworth Inc, Belmont California, 1984.

[6] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF Regional conference series in applied mathematics, 1992.

[7] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon university, 1988.

[8] M.A. Franzini. Speech recognition with back-propagation. In *Proceedings, Ninth Annual Conference of IEEE Engineering in Medicine and Biology Society*, 1987.

[9] C. Gégout, B. Girau, and F. Rossi. A general feedforward neural network model. Technical report NC-TR-95-041, NeuroCOLT, Royal Holloway, University of London, 1995.

[10] C. Gégout, B. Girau, and F. Rossi. Generic back-propagation in arbitrary feedforward neural networks. In *Artificial Neural Nets and Genetic Algorithms - Proc. of ICANNGA*, pages 168–171. Springer-Verlag, 1995.

[11] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[12] R.A. Jacobs. Increased rates of convergence through learning rate adaptatio. *Neural Networks*, 1(4):295–307, 1998.

[13] Y. LeCun, P.Y. Simard, and B. Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In Morgan Kaufmann, editor, *Advances in neural information processing systems*, volume 5, pages 156–163, 1993.

[14] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis. Effective backpropagation training with variable stepsize. *Neural Networks*, 10(1):69–82, 1997.

[15] K. Myung-Chan and C. Chong-Ho. A new weight initialization method for the MLP with the BP in multiclass classification problems. *Neural Processing Letters*, 6:11–23, 1997.

[16] D.B. Parker. Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order hebbian learning. In *Proc. First Int. Joint Conf. Neural Networks*, volume II, pages 593–600, Nov. 1987.

[17] D.C. Plaut, S.J. Nowlan, and G.E. Hinton. Experiments on learning by back-propagation. Technical Report CMU-CS-86-126, Carnegie Mellon university, 1986.

[18] G. Thimm, P. Moerland, and E. Fiesler. The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation*, 8:451–460, 1996.

[19] R.L. Watrous. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optmization. In *Proc. IEEE First Int. Conf. Neural Networks*, volume 2, pages 619–626, June 1987.

[20] Q. Zhang and A. Benveniste. Wavelet networks. *IEEE Trans. On Neural Networks*, 3(6):889–898, Nov. 1992.