Neural Processing Letters **7:** 5–14, 1998.

Constrained Learning in Neural Networks: Application to Stable Factorization of 2-D Polynomials

STAVROS PERANTONIS[1], NIKOLAOS AMPAZIS[1],
STAVROS VAROUFAKIS[1] and GEORGE ANTONIOU[2]
[1]*Institute of Informatics and Telecommunications, National Center for Scientific Research 'Demokritos', 153 10 Aghia Paraskevi, Athens, Greece;* [2]*Department of Mathematics and Computer Science, Montclair State University, Montclair, New Jersey 07043, USA*
E-mail: sper@iit.nrcps.ariadne-t.gr; antonioug@alpha.montclair.edu

Key words: constrained learning, factorization, feedforward networks, IIR filters, polynomials, stability

Abstract. Adaptive artificial neural network techniques are introduced and applied to the factorization of 2-D second order polynomials. The proposed neural network is trained using a constrained learning algorithm that achieves minimization of the usual mean square error criterion along with simultaneous satisfaction of multiple equality and inequality constraints between the polynomial coefficients. Using this method, we are able to obtain good approximate solutions for non-factorable polynomials. By incorporating stability constraints into the formalism, our method can be successfully used for the realization of stable 2-D second order IIR filters in cascade form.

1. Introduction

Although a central theme in neural network research has been learning by the sole use of examples, recent investigations have revealed the role of augmenting the learning process by taking into account *a priori* available network specific or problem specific knowledge [1]. Various techniques have been adopted for incorporating such knowledge into the network architecture [2–4] or learning rule [5] and thus enhancing network performance.

In this work, we present a constrained learning formalism in order to exploit problem-specific information in the form of exact or approximate constraints imposed upon the weights. The specific application that will be addressed using this formalism is multidimensional polynomial factorization, an important problem with applications in multidimensional signal processing, mathematics and mathematical physics. It is a difficult problem because the fundamental theorem of Algebra does not apply to multidimensional polynomials, and techniques applicable to 1-D polynomials cannot be generalized to many dimensions.

Many methods and algorithms have been proposed for dealing with polynomial factorization [6–8]. Recently, neural networks have been applied to the related, but considerably easier problem of separating multidimensional polynomials into a product of 1-D polynomials [9]. The method introduced in this work will be applied to the factorization of 2-D, second order polynomials and is especially useful for providing good approximate solutions for non-factorable polynomials.

We first discuss the application of back propagation (BP) to the factorization problem. Next, we introduce an improved 'constrained learning' algorithm (CLA), which takes into account *a priori* available information encoded in multiple equality or inequality constraints between the weights, reflecting relations between the coefficients of the desired factor polynomials.

Using the CLA technique, we incorporate two classes of information for the polynomial factorization problem. The first class involves constraints between the coefficients of the original polynomial and those of the factor polynomials. Inclusion of these constraints leads to improved evaluation of the desired factor polynomials in comparison to BP. The second class of constraints is related to the design of IIR filters. Factorization is useful for realizing this type of filters in cascade form. This is a common method due to its relative insensitivity to coefficient quantization. In this case, stability of the factor polynomials is very important and should be a prime concern of any factorization method. Thus, we use additional inequality constraints in order to ensure the stability (in the sense of Huang [11]) of the factor polynomials. The proposed method is superior to BP, as it leads more often to the stability of the final polynomial product.

2. Network architecture and back propagation

Consider the polynomial

$$f(z_1, z_2) = \sum_{i=0}^{2} \sum_{j=0}^{2} a_{ij}\, z_1^i\, z_2^j \;\text{ with } a_{00} = 1 \tag{1}$$

We seek to achieve an exact or approximate factorization of the form:

$$f(z_1, z_2) \approx \prod_{i=1,2} h_i(z_1, z_2),\; h_i(z_1, z_2) = 1 + w_{i1} z_1 + w_{i2} z_2 + w_{i3} z_1 z_2 \tag{2}$$

To this end, we utilize the neural network shown in Figure 1. The units in the hidden layer have a logarithmic activation function $f(x) = \ln(|x|)$, whereas the output unit is linear. This architecture has been preferred to the Sigma-Pi architecture employed in [9], because it produces smoother cost function landscapes avoiding deep valleys and thus facilitates learning. Thus, the network's outputs are expressed in terms of its weights as follows:

$$\begin{aligned} O_p \;&=\; \ln|1 + w_{11} z_{1,p} + w_{12} z_{2,p} + w_{13} z_{1,p} z_{2,p}| \\ &+\; \ln|1 + w_{21} z_{1,p} + w_{22} z_{2,p} + w_{23} z_{1,p} z_{2,p}| \end{aligned} \tag{3}$$

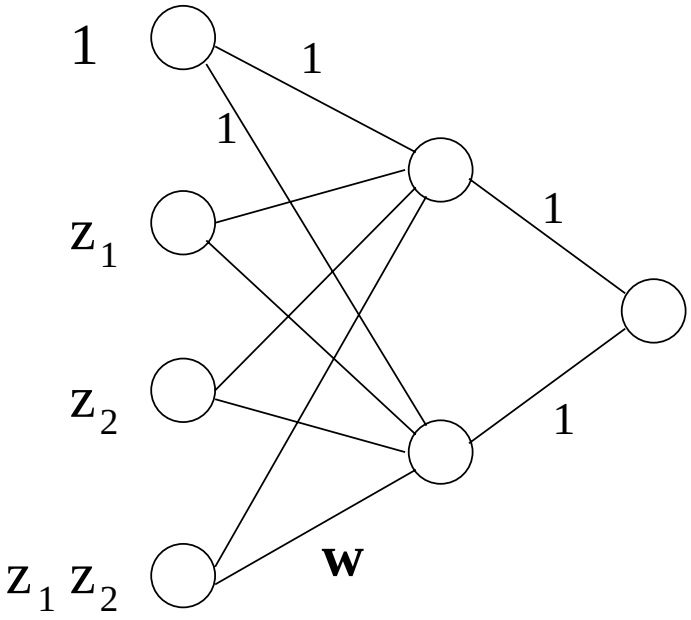

Figure 1. Neural network architecture used to solve the factorization problem.

Learning will proceed using P training patterns, which are selected from the region $|z_1| < 1$, $|z_2| < 1$. The purpose of the learning rule is to minimize the mean square error:

$$E = \frac{1}{2}\sum_{p=1}^{P}(T_p - O_p)^2, \;\; T_p = \ln|f(z_{1,p}, z_{2,p})| \tag{4}$$

The gradient descent (BP) weight update rule can be written: $dw_{ij} = -\eta J_{ij}$ where dw_{ij} is the difference between updated and current epoch weight, and the 2×3 matrix J is defined as:

$$J = \sum_p (O_p - T_p)\begin{bmatrix} \frac{z_{1,p}}{|h_1(z_{1,p},z_{2,p})|} & \frac{z_{2,p}}{|h_1(z_{1,p},z_{2,p})|} & \frac{z_{1,p}z_{2,p}}{|h_1(z_{1,p},z_{2,p})|} \\ \frac{z_{1,p}}{|h_2(z_{1,p},z_{2,p})|} & \frac{z_{2,p}}{|h_2(z_{1,p},z_{2,p})|} & \frac{z_{1,p}z_{2,p}}{|h_2(z_{1,p},z_{2,p})|} \end{bmatrix} \tag{5}$$

3. Constrained learning: Equality constraints

Additional information is available in the form of equality constraints involving the weights. These reflect the conditions between the coefficients of the given polynomial and the coefficients of the desired factor polynomials. There are eight constraints in all, but best results have been obtained by using only constraints involving the coefficients of first order terms in z_1 and z_2:

$$\begin{aligned} \Phi_1 &= a_{10} - w_{11} - w_{21} = 0, \quad \Phi_2 = a_{01} - w_{12} - w_{22} = 0 \\ \Phi_3 &= a_{11} - w_{11}w_{22} - w_{12}w_{21} - w_{13} - w_{23} = 0 \end{aligned} \tag{6}$$

It is very important to note that it is not desirable to hardwire such constraints into the network architecture, since we want to be able to provide approximate factorizations to non-factorable polynomials. Instead, we want to construct a learning algorithm that can progressively move the Φ_i as close as possible to their targets (zero).

To take account of the constraints, we modify the objective of our learning process. The new objective will be to reach a minimum of the cost function in (4) with respect to the variables w_{ij}, which fulfills as best as possible the constraints $\boldsymbol{\Phi} = \mathbf{0}$, where $\boldsymbol{\Phi} = (\Phi_1, \Phi_2, \Phi_3)$.

The strategy which be followed to solve this problem is an extention, for the case of multiple constraints, of the method used by Perantonis and Karras for the efficient training of multilayered feedforward networks with just one equality constraint [5, 10]. At each epoch of the learning process, the synaptic weights will be changed by small amounts dw_{ij}, so that

$$\sum_{ij} (dw_{ij})^2 = (\delta P)^2 \tag{7}$$

where δP is a constant. If δP is small enough, the changes to E and to $\boldsymbol{\Phi}$ induced by changes in the weights can be approximated by the first differentials dE and $d\boldsymbol{\Phi}$.

At each epoch, we seek to achieve the maximum possible change in $|dE|$ (minimum in $dE < 0$), so that (7) is respected, and the change $d\boldsymbol{\Phi}$ in $\boldsymbol{\Phi}$ is equal to a predetermined vector quantity $\delta \boldsymbol{Q}$, designed to bring $\boldsymbol{\Phi}$ closer to its target (zero). This is a constrained optimization problem which can be solved analytically by introducing a vector $\boldsymbol{\lambda} = (\lambda_m, m = 1, \ldots, 3)$ of Lagrange multipliers to take account of the constraints in (6) and a further Lagrange multiplier μ to take account of (7). Upon solving the optimization problem, we are led to the following equations for the Lagrange multipliers:

$$\mu = -\frac{1}{2} \left[\frac{I_{JJ} - \boldsymbol{I}_{JF}^T \boldsymbol{I}_{FF}^{-1} \boldsymbol{I}_{JF}}{(dP)^2 - \delta \boldsymbol{Q}^T \boldsymbol{I}_{FF}^{-1} \delta \boldsymbol{Q}} \right]^{1/2} \tag{8}$$

$$\boldsymbol{\lambda} = -2\mu \boldsymbol{I}_{FF}^{-1} \delta \boldsymbol{Q} + \boldsymbol{I}_{FF}^{-1} \boldsymbol{I}_{JF} \tag{9}$$

where I_{JJ} (a scalar), $\boldsymbol{I}_{JF}$ (a 3-component vector) and $\boldsymbol{I}_{FF}$ (a 3×3 matrix) are defined by:

$$\begin{array}{l} I_{JJ} = \sum_{ij} \left(J_{ij}\right)^2, \\ I_{JF}^m = \sum_{ij} J_{ij} F_{ij}^m, \;\; I_{FF}^{mn} = \sum_{ij} F_{ij}^m F_{ij}^n, \;\; F_{ij}^m = \partial \Phi_m / \partial w_{ij} \end{array} \tag{10}$$

In terms of the quantities defined above, the weight update rule for CLA becomes:

$$dw_{ij} = \frac{J_{ij}}{2\mu} - \frac{\boldsymbol{F}_{ij}^T \boldsymbol{\lambda}}{2\mu} \tag{11}$$

A full derivation is given in the appendix. See also references [12], [5] and [10].

We next discuss the choice of the quantities δQ_m. It is natural to choose δQ_m proportional to Φ_m with a negative constant of proportionality $-k, k > 0$, so that the constraints move towards zero at an exponential rate. With this choice for k, note the bound $k \leq dP \left(\boldsymbol{\Phi}^T \boldsymbol{I}_{FF}^{-1} \boldsymbol{\Phi} \right)^{-1/2}$ set on the value of k by (**??**), which forces us to choose k adaptively. The simplest choice for k will be used, namely $k = \left[\xi (dP)^2 / (\boldsymbol{\Phi}^T \boldsymbol{I}_{FF}^{-1} \boldsymbol{\Phi}) \right]^{1/2}$, where $0 < \xi < 1$ is a free parameter of the algorithm. Thus, the final CLA weight update rule has only two free parameters, namely dP and ξ.

4. Inequality constraints

Consider the IIR filter $1/f(z_1, z_2)$ with $f(z_1, z_2)$ defined as in (1). The stability conditions according to Huang [11] for the factor polynomials can be written as $X_k < 1, k = 1, \ldots, 6$, where

$$\begin{aligned} X_1 &= w_{11}^2, \quad X_2 = w_{21}^2, \quad X_3 = w_{12}^2 + w_{13}^2 + 2w_{12}w_{13} - w_{11}^2 - 2w_{11}, \\ X_4 &= w_{12}^2 + w_{13}^2 - 2w_{12}w_{13} - w_{11}^2 + 2w_{11}, \\ X_5 &= w_{22}^2 + w_{23}^2 + 2w_{22}w_{23} - w_{21}^2 - 2w_{21}, \\ X_6 &= w_{22}^2 + w_{23}^2 - 2w_{22}w_{23} - w_{21}^2 + 2w_{21} \end{aligned} \tag{12}$$

To incorporate such inequality conditions into CLA, we proceed as follows: At a certain epoch of the learning process, let a number M of the stability constraints be violated, so that $X_{k_r} \geq 1, r = 1, \ldots, M$. In this case, we can augment the vector $\boldsymbol{\Phi}$ of the previous section by adding extra components $\Phi_{3+r} = X_{k_r}, r = 1, \ldots M$. By following the formalism of the previous section using the augmented vector $\boldsymbol{\Phi}$, the quantities X_{k_r} are driven towards the stability region. Note that the dimensionality of the vector $\boldsymbol{\Phi}$ is adaptive, as only the violated constraints are included in each epoch. Iterative application of this scheme can lead to a final stable solution.

In practice, to make sure that the final solution does not end up very close to the border of the stability region, a small safety margin $\delta > 0$ can be employed, so that addition of an extra component to the vector $\boldsymbol{\Phi}$ is made whenever $X_{k_r} \geq 1 - \delta$. In our simulations, the value $\delta = 10^{-2}$ was used.

Note that a necessary (but not sufficient) condition for stability is $|w_{ij}| < 1$. Therefore, standard BP initialized with relatively small initial weights often leads to stable solutions. However, cases where back propagation fails can be tackled more effectively using CLA, as illustrated in the next section.

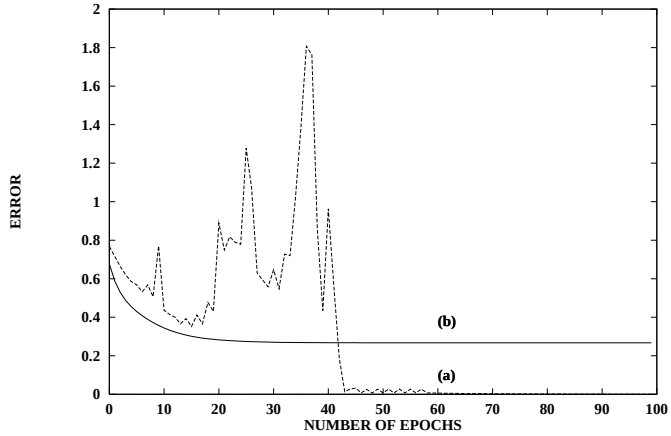

Figure 2. Error curves for (a) CLA and (b) BP.

5. Simulations

Example 1: Exactly factorable polynomial. A neural network is employed to factorize a polynomial $f(z_1, z_2)$ of the form given by (1) with

$$\boldsymbol{a} = \begin{bmatrix} 1 & 1.5 & 0.5 \\ 4 & 5 & 2.25 \\ 3 & 6.5 & 1 \end{bmatrix}$$

This polynomial is exactly factorable to:

$$f = (1 + z_1 + 0.5z_2 + 2z_1z_2)(1 + 3z_1 + z_2 + 0.5z_1z_2)$$

A training set of $P = 30$ patterns was formed. For BP learning we used the value $\eta = 10^{-2}$ for the learning rate, while for CLA we used the values $\delta P = 0.03$, $\xi = 0.09$ for the learning parameters. Figure 2 shows the error curves for neural networks trained using standard BP and CLA respectively. Evidently, BP learning reaches an undesirable flat local minimum after a few epochs and becomes practically inoperable, while CLA overcomes the difficulty, finds the global minimum and gives the following factorization result:

$$\begin{aligned} f_{CLA} \approx\ & (1+0.9991z_1+0.5004z_2+2.0011z_1z_2) \\ & (1+3.0009z_1+0.9996z_2+0.5029z_1z_2) \end{aligned}$$

which compares favorably to the exact factorization result.

Example 2: Approximate factorization. Consider the (non-factorable) polynomial $f(z_1, z_2)$ with

$$\boldsymbol{a} = \begin{bmatrix} 1 & -1.2 & 0.5 \\ -1.5 & 1.8 & -0.75 \\ 0.6 & 0.72 & 0.29 \end{bmatrix}$$

This polynomial is quoted in [13]. Interestingly, we have found good approximate factorizations to this polynomial using our methods. A training set of $P = 100$ patterns was formed. For BP learning we used the value $\eta = 8 \cdot 10^{-4}$ for the learning rate, while for CLA we used the values $\delta P = 0.01$, $\xi = 0.9$ for the learning parameters. Both BP and CLA have reached approximate solutions to the factorization problem:

$$\begin{aligned} f_{CLA} &\approx (1-0.7780z_1-0.6572z_2+0.5529z_1z_2) \\ &\quad\ (1-0.7220z_1-0.5428z_2+0.3606z_1z_2) \\ f_{BP} &\approx (1-0.6843z_1-0.4727z_2+0.2475z_1z_2) \\ &\quad\ (1-0.7376z_1-0.5362z_2+0.4371z_1z_2) \end{aligned}$$

The corresponding estimates for the original polynomial coefficients are:

$$\boldsymbol{a}_{CLA} = \begin{bmatrix} 1 & -1.200 & 0.357 \\ -1.500 & 1.810 & -0.537 \\ 0.562 & -0.680 & 0.199 \end{bmatrix} \quad \boldsymbol{a}_{BP} = \begin{bmatrix} 1 & -1.009 & 0.253 \\ -1.420 & 1.401 & -0.340 \\ 0.503 & -0.482 & 0.108 \end{bmatrix}$$

CLA has been able to provide more accurate estimates to the original polynomial coefficients, thanks to the additional information about the constraints involving these coefficients.

Example 3: Stable approximate factorization. Consider the IIR filter defined by $1/f(z_1, z_2)$, where $f(z_1, z_2)$ is of the form given by (1) with

$$\boldsymbol{a} = \begin{bmatrix} 1 & -1.0 & 0.3 \\ -0.768 & 0.61 & -0.13 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

The same training set size and learning parameters were used as in example 2. By incorporating the additional constraints of section 2 and the stability constraints of section 3 into CLA, we are able to obtain the following approximate factorization:

$$\begin{aligned} f_{CLA} \approx\ & (1-0.7659z_1-0.4372z_2+0.2268z_1z_2) \\ & (1-0.0027z_1-0.5632z_2-0.0350z_1z_2) \end{aligned}$$

The BP algorithm yields the factorization:

$$\begin{aligned} f_{BP} \approx\ & (1-0.7084z_1-0.4628z_2+0.1253z_1z_2) \\ & (1-0.0820z_1-0.4707z_2+0.1769z_1z_2) \end{aligned}$$

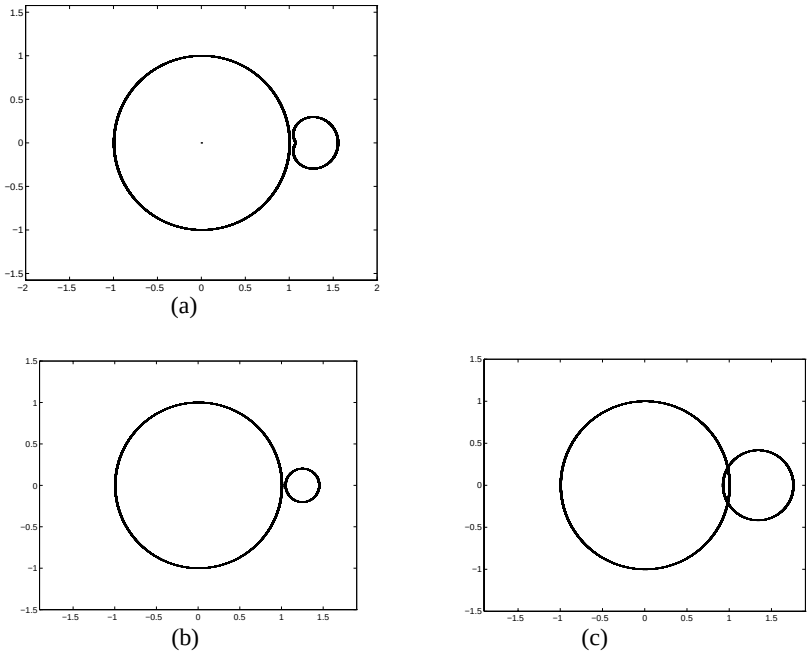

Figure 3. Root maps for the IIR filters. (a) Original filter (b) Filter obtained by CLA (c) Filter obtained by BP.

The corresponding estimates for the coefficients of the original polynomial are:

$$\boldsymbol{a}_{CLA} = \begin{bmatrix} 1 & -1.000 & 0.246 \\ -0.769 & 0.624 & -0.112 \\ 0.002 & 0.026 & -0.008 \end{bmatrix} \quad \boldsymbol{a}_{BP} = \begin{bmatrix} 1 & -0.933 & 0.218 \\ -0.790 & 0.673 & -0.141 \\ 0.058 & -0.136 & 0.022 \end{bmatrix}$$

Both factor polynomials obtained through CLA satisfy Huang's conditions and the cascade filter thus designed is stable. By contrast, the first factor polynomial obtained by BP does not satisfy Huang's stability condition and the corresponding IIR filter is unstable. The root map [13] for the original polynomial is shown in Figure 3a. The two contours, corresponding to $|z_1| = 1$ and $|z_2| = 1$, do not overlap, as a result of the original IIR filter stability. In Figures 3b and 3c, root maps are shown for the polynomial products respectively obtained by CLA and BP. In Figure 3b, the contours do not overlap as a result of IIR filter stability. By contrast, the overlapping of contours in Figure 3c signifies instability of the filter obtained by BP.

6. Conclusion and Prospects

In this work, adaptive neural network based techniques have been proposed for the factorization of 2-D, second order polynomials. Using a flexible framework for

incorporating additional information in the form of constraints, we obtained exact solutions for factorable polynomials and, more importantly, good approximate solutions for non-factorable polynomials. A principal achievement of the method is the ability to factorize 2-D polynomials, preserving the stability criteria at the same time. Thus, stable factorizations for IIR filters have been achieved using our method.

The neural network architecture described in this work can be easily expanded, so that factorization of higher order polynomials can be studied in detail. Moreover, the role of neural network weight elimination and pruning techniques should be investigated, as these methods could prove very useful for identifying missing polynomial coefficients (cf. example 3 above) and obtaining robust factorizations.

Appendix

We introduce the function ε, whose differential is defined as follows:

$$d\,\varepsilon = dE + (\delta \boldsymbol{Q}^T - d\boldsymbol{\Phi}^T)\boldsymbol{\lambda} + \mu[(dP)^2 - \sum_{ij}(dw_{ij})^2] \tag{13}$$

On evaluating the differentials involved in the right hand side, we readily obtain:

$$d\varepsilon = \sum_{ij} J_{ij} dw_{ij} + (\delta \boldsymbol{Q}^T - \sum_{ij} \boldsymbol{F}_{ij}^T dw_{ij})\boldsymbol{\lambda} + \mu[(dP)^2 - \sum_{ij}(dw_{ij})^2] \tag{14}$$

To minimize $d\varepsilon$ (maximize its magnitude) at each epoch, we demand that:

$$\begin{array}{rcl} d^2\varepsilon & = & \sum_{ij}\left(J_{ij} - \boldsymbol{F}_{ij}^T\boldsymbol{\lambda} - 2\mu dw_{ij}\right) d^2 w_{ij} = 0, \\ d^3\varepsilon & = & -2\mu \sum_{ij}(d^2 w_{ij})^2 > 0 \end{array} \tag{15}$$

Hence, the coefficients of $d^2 w_{ij}$ in the last equation should vanish, and therefore (11) holds. Equation (11) constitutes the weight update rule for the neural network, provided that μ and $\boldsymbol{\lambda}$ be evaluated in terms of known quantities. This can be done as follows: Noting that $d\boldsymbol{\Phi} = \delta \boldsymbol{Q}$ and taking (**??**) into account, we obtain:

$$\delta \boldsymbol{Q} = \frac{1}{2\mu}(\boldsymbol{I}_{JF} - \boldsymbol{I}_{FF}^T\boldsymbol{\lambda}) \tag{16}$$

with $\boldsymbol{I}_{JF}$ and $\boldsymbol{I}_{FF}$ defined as in (10). Equation (16) can be readily solved for $\boldsymbol{\lambda}$, whereupon (9) is deduced.

It remains to evaluate μ. To this end, we substitute (11) into (7):

$$4\mu^2(dP)^2 = I_{JJ} + \boldsymbol{\lambda}^T\boldsymbol{I}_{FF}\boldsymbol{\lambda} - \boldsymbol{I}_{JF}^T\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\boldsymbol{I}_{JF} \tag{17}$$

where I_{JJ} is defined in (10). Finally, we substitute (9) into (17) and solve for μ to obtain (8). Note that the negative square root value has been chosen for μ in order to satisfy the second of equations (15).

References

1. D. Barber and D. Saad, "Does extra knowledge necessarily improve generalization?", Neural Computation, Vol. 8, pp. 202–214, 1996.
2. Y. le Cun, L.D. Jackel, B.E. Boser, J.S. Denker, H-P. Graf, I.Guyon, D. Henderson, R.E. Howard and W.Hubbard, "Handwritten digit recognition: Applications of neural network chips and automatic learning", IEEE Communications Magazine, pp. 41–46, November 1989.
3. P. Simard, Y. le Cun and J. Denker, "Efficient pattern recognition using a new transformation distance", in S.J. Hanson, J.D. Cowan and C.L. Giles (eds) Advances in Neural Processing Systems, Vol. 5, pp. 50–58, Morgan Kaufmann, San Mateo, 1993.
4. S. Gold, A. Rangarajan and E. Mjolsness, "Learning with preknowledge: clustering with point and graph matching distance", Neural Computatation, Vol. 8, pp. 787–804, 1996.
5. S.J. Perantonis and D.A. Karras, "An efficient constrained learning algorithm with momentum acceleration", Neural Networks, Vol. 8, pp. 237–239, 1995.
6. Z. Mou-Yan and R. Unbehauen, "On the approximate factorization of 2-D polynomials", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-35, pp. 577–579, 1987.
7. N.E. Mastorakis, N.J. Theodorou and S.G. Tzafestas, "A general factorization method for multivariable polynomials", Multidimensional Systems and Signal Processing, Vol. 5, pp. 151–178, 1994.
8. P. Misra and R.V. Patel, "Simple factorizability of 2-D polynomials", in Proc. Int. Symp. Circ. Syst., pp. 1207–1210, New Orleans, 1990.
9. R. Hormis, G. Antoniou and S. Mentzelopoulou, "Separation of two-dimensional polynomials via a sigma-pi neural net", in Proc. Int. Conf. on Modelling and Simulation, pp. 304–306, Pittsburg, PA, USA, 1995.
10. D.A. Karras and S.J. Perantonis, "An efficient constrained training algorithm for feedforward networks", IEEE Trans. Neural Networks, Vol. 6, pp. 1420–1434, 1995.
11. T.S. Huang, "Stability of two-dimensional recursive filters", IEEE Trans. Audio Electroacoust., Vol. AU-20, pp. 158–163, 1972.
12. A.E. Bryson and W.F. Denham, "A steepest-ascent method for solving optimum programming problems", J. Appl. Mechanics, Vol. 29, pp. 247–257, 1962.
13. J.L. Shanks, S. Treitel and J.H. Justice, "Stability and synthesis of two-dimensional recursive filters", IEEE Trans. Audio Electroacoust., Vol. AU-20, pp. 115–128, 1972.