Direct Parallel Perceptrons (DPPs): Fast Analytical Calculation of the Parallel Perceptrons Weights with Margin Control for Classification Tasks

Manuel Fernández-Delgado, Jorge Ribeiro, Eva Cernadas, and Senén Barro Ameneiro

***Abstract*— Parallel perceptrons (PPs) are very simple and efficient committee machines (a single layer of perceptrons with threshold activation functions and binary outputs, and a majority voting decision scheme), which nevertheless behave as universal approximators. The parallel delta (P-Delta) rule is an effective training algorithm, which, following the ideas of statistical learning theory used by the support vector machine (SVM), raises its generalization ability by maximizing the difference between the perceptron activations for the training patterns and the activation threshold (which corresponds to the separating hyperplane). In this paper, we propose an analytical closed-form expression to calculate the PPs' weights for classification tasks. Our method, called *Direct Parallel Perceptrons* (DPPs), directly calculates (without iterations) the weights using the training patterns and their desired outputs, without any search or numeric function optimization. The calculated weights globally minimize an error function which simultaneously takes into account the training error and the classification margin. Given its analytical and noniterative nature, DPPs are computationally much more efficient than other related approaches (P-Delta and SVM), and its computational complexity is linear in the input dimensionality. Therefore, DPPs are very appealing, in terms of time complexity and memory consumption, and are very easy to use for high-dimensional classification tasks. On real benchmark datasets with two and multiple classes, DPPs are competitive with SVM and other approaches but they also allow *online* learning and, as opposed to most of them, have no tunable parameters.**

***Index Terms*— Analytical closed-form weight calculation, linear computational complexity, margin maximization, online learning, parallel delta rule, parallel perceptrons, pattern classification.**

Manuscript received June 22, 2009; revised September 5, 2011; accepted September 12, 2011. Date of publication October 6, 2011; date of current version November 2, 2011. This work was supported in part by Xunta de Galicia under Project 08MMA010402 PR and the Spanish Ministry of Science and Innovation (MICINN) under Project TIN2009-07737/TIN2009-14372-C03-03.

M. Fernández-Delgado and S. B. Ameneiro are with the Intelligent Systems Group, Gipuzkoa 20018, Spain. They are also with the Centro de Investigación en Tecnoloxías da Información da USC (CITIUS), University of Santiago de Compostela, Santiago de Compostela CP 15782, Spain (e-mail: manuel.fernandez.delgado@usc.es; senen.barro@usc.es).

J. Ribeiro is with the School of Technology and Management, Viana do Castelo Polytechnic Institute, Viana do Castelo 4900-348, Portugal (e-mail: jribeiro@estg.ipvc.pt).

E. Cernadas is with the Centro de Investigación en Tecnoloxías da Información da USC (CITIUS), University of Santiago de Compostela, Santiago de Compostela CP 15782, Spain (e-mail: eva.cernadas@usc.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2011.2169086

I. Introduction

PARALLEL perceptrons (PPs) [1], [2] are extremely simple neural networks consisting in a layer of n single perceptrons with threshold activation functions and binary outputs $\{\pm 1\}$, connected to a *soft-winner-take-all* gate (SWTA) [3], which implements a voting scheme. The relevance of [2] is that: 1) it proves that PPs are universal approximators, and 2) it proposes an effective training algorithm (the parallel delta or P-Delta rule) which provides performance not much different from more powerful models without biological plausibility. This issue is very important, because for a long time it was thought that there was no competitive learning algorithm for these simple computing structures (which can be considered as committee machines).

The P-Delta rule iteratively updates the weights $\mathbf{w}$ of the single perceptrons attempting simultaneously to classify correctly the training patterns $\mathbf{x}$ (although errors are allowed) while incrementing the distance between the patterns and the hyperplanes of the perceptrons, in order to avoid that small changes in the input (due to noise or pattern variability) may cause the patterns to be assigned to the other class. In order to achieve this objective, the weight $\mathbf{w}_i$ of the perceptron i is updated not only if $\mathbf{w}_i^T\mathbf{x}$ has the wrong sign (the updating uses the classical delta rule), but also if $|\mathbf{w}_i^T\mathbf{x}| < \gamma$, where γ is a parameter which defines the lowest required margin.

PPs trained with the P-Delta rule were compared in [2] with other state-of-the-art classifiers, including the Multilayer Perceptron (MLP), Madaline, Decision Tree (C4.5), and Support Vector Machine (SVM), providing comparable results despite its simplicity. Although the P-Delta rule was proposed for general learning problems (classification, regression, etc.), in [2] it was applied only for classification with two classes. Compared to other approaches, PPs are not very sensible with respect to their tunable parameters, such as the required margin γ and the number n of perceptrons, among others. Studies on the activation margin of the PPs with imbalanced training sets [4] and with noisy labels [5] have been published recently. In [6], a balanced boosting of PPs is proposed, which uses the PPs margins in order to find incorrectly labeled and redundant training patterns. In [7], another network with real-valued trainable output weights—as opposed to the fixed connections in the original PPs—is proposed to be computed by applying linear discriminant analysis (LDA) to the perceptron outputs. Although not related to the original model in [2], a network

of PPs was also proposed in [8] to solve a multiclass problem by dividing it into several two-class problems.

The simplicity of the PPs prompted us to search for an analytical expression to directly calculate the perceptron weights without any iterative training for classification problems (with two and multiple classes). The objective of this paper is to derive an analytical closed-form expression for the PPs weights using only the training patterns $\{\mathbf{x}_k\}_{k=1}^N$ and their desired outputs $\{d_k\}_{k=1}^N$ (class labels), without any training process involving numerical iterative calculations, which are often conditioned by training problems (local minima, oscillations) and parameter tuning (number of hidden neurons, learning rate, momentum, weight initialization, etc.). Our approach, called *Direct Parallel Perceptrons* (DPPs), minimizes the training error and maximizes the classification margin, using the ideas of statistical learning theory [9] such as SVM and P-Delta. However, DPPs analytically calculate the weights that minimize the error, while P-Delta uses a slower iterative updating procedure. We will see that DPPs are computationally much more efficient than the standard iterative methods, because the weights are calculated directly using the training patterns without iterations and convergence problems, without parameter tuning and getting stuck in local minima, and without expensive matrix or geometric calculations which do not scale well with the input dimensionality. Formulations of DPPs for classification problems with two and more classes are presented in this paper, achieving competitive classification accuracies with the state-of-the-art classifiers for many datasets.

This paper is organized as follows. In Section II, we compile the approaches related to DPPs in the literature. Section III-A briefly resumes the PP network and the P-Delta rule. The analytical calculation of the single perceptron weights is described in Section III-B. The DPPs are presented for two-class problems in Section III-C (the whole method is compiled in Section III-D) and for multiclass problems in Section III-E. Section IV compares the results of DPPs, P-Delta, and other classifiers using benchmark two-class (Section IV-A) and multiclass (Section IV-B) problems, and evaluates the execution times (Section IV-C). Section V discusses the results and Section VI reports the conclusions of this paper and the future research.

II. Comparison with Existing Approaches

Although some research works train perceptron networks by maximizing margin (e.g., the Margitron [10]), and others even train SVMs with perceptron-like algorithms (SoftDoubleMaxMinOver [11]), there are not many approaches that directly calculate the weights in the perceptron-based networks, because their topological complexity and the perceptron activation functions lead to mathematical expressions that are not analytically treatable. Among the approaches referred to in the previous section, only [7] modifies the original PP model, but it does not use a closed-form expression for the weights. In [12], a closed-form neural network is proposed for the extraction of maximum discriminatory features in high-dimensional data using neurons with polynomial activation functions. The polynomial coefficients are calculated in order to minimize a measure of the discrimination between two classes. In [13], a closed-form solution for a kernel-based classifier is used with the same output as an SVM. The coefficients of the kernel applied to each training pattern are calculated using the Moore–Penrose pseudo-inverse of an energy matrix for each class. This matrix is computed to maximize a criterion to discriminate between classes. However, despite of the closed-form solution, its computational cost is very high, so [14] recommends the use of an adaptive method. Besides, this approach was tested only on the "Handwritten digits" dataset [15].

Another remarkable proposal is given in [16], which directly calculates the weights of a standard two-class MLP network using a geometric method to design the weights which map the input patterns to the feature space, where the linear separability of the patterns is guaranteed. With more than two classes, several two-class MLPs (of the kind proposed in this paper) must be trained, instead of a standard multiclass MLP using a similar approach. This issue might raise its computational complexity. On the other hand, this method is based on the iterative calculation of the polyhedron minimal volume of a set of points, whose determination might be computationally expensive, especially for high-dimensional datasets with many classes.

Piecewise linear classifiers (PLCs) are also approaches that directly calculate the hyperplane equations using geometric procedures. The PLC in [17] uses a set of clusters on the training set and builds the set of Tomek links [18] between pairs of prototypes belonging to different classes. Then the hyperplanes that cut all the links are found, and the majority rule is used to assign each region surrounded by these hyperplanes to a class. The prototypes are well classified, but there is no guarantee for the remaining training patterns. Again, their computational cost might be large at high input dimensions, because the method requires many geometric calculations in order to find the Tomek links and the hyperplanes. Finally, the accuracy depends on the clustering quality, and the PLC often fails near the class boundaries. Previous PLC approaches exist (see a brief review in [17]), but they require estimations of the probability functions, use strong restrictions on the class shapes, or have a high computational cost. Recently, a set of PLCs (the multiconlitron [19]), with a solid geometric foundation, has been proposed. It learns the set of hyperplanes that surround a convex region, separating two convexly separable classes. However, this convexity constraint may restrict its performance, which is below Gaussian kernel SVM for some datasets.

III. PP and DPP

Before the description of our proposal (DPPs) for the classification with two and more classes, we will briefly compile the main features of the PP network and the P-Delta rule. We will use the nomenclature given in Table I.

A. PP and Parallel Delta Rule

A PP network (for two-class classification problems) [2] is composed of a layer of n single perceptrons with outputs

TABLE I
NOTATION USED IN THE TEXT

d	Dimension of the input space
N	Number of training patterns
N_c	Number of classes
γ	Desired margin
$\mathbf{x}_k$	Training pattern k ($k = 1, \ldots, N$)
d_k	Desired output for the PPs and training pattern $\mathbf{x}_k$
Single perceptron	
$\mathbf{w}$	Weight vector ($\lvert\mathbf{w}\rvert = 1$) of the perceptron
y	Perceptron output
Two-class parallel perceptrons (n perceptrons)	
n	Number of perceptrons in the PPs (it must be odd)
$\mathbf{w}_i$	Weight vector ($\lvert\mathbf{w}_i\rvert = 1$) of perceptron i ($i = 1, \ldots, n$)
y_i	Output of the perceptron i
d_{ik}	Desired output for the perceptron i and training pattern $\mathbf{x}_k$
y	Output of the PPs
Multiclass PP (N_c two-class PP)	
n_l	Number of perceptrons in the PPs l ($l = 1, \ldots, N_c$)
$\mathbf{w}_{li}$	Weight vector ($\lvert\mathbf{w}_{li}\rvert = 1$) of perceptron i ($i = 1, \ldots, n_l$) in the PPs l
y_{li}	Output of the perceptron i in the PPs l
d_{lik}	Desired output for the perceptron i in the PPs l and training pattern $\mathbf{x}_k$
y_l	Output of the PPs l
d_{lk}	Desired output for the PPs l and training pattern $\mathbf{x}_k$

$y_i(\mathbf{x}), i = 1, \ldots, n$, connected to a SWTA gate with output $y(\mathbf{x})$. Given an input pattern $\mathbf{x} \in \mathbb{R}^d$, the output $y_i(\mathbf{x})$ of the perceptron i is given by

$$y_i(\mathbf{x}) = \text{sign}\left(\mathbf{w}_i^T\mathbf{x}\right), \qquad i = 1, \ldots, n. \tag{1}$$

We denote $\mathbf{w}_i^T$ the transposed of vector $\mathbf{w}_i$ (it must be $\lvert\mathbf{w}_i\rvert = 1$), and $\mathbf{w}_i^T\mathbf{x}$ is the dot product (a constant bias input or offset $x_d = 1$ is assumed) of $\mathbf{w}_i$ and $\mathbf{x}$. The PPs output is given by $y(\mathbf{x}) = \text{sign}\left[\sum_{i=1}^n y_i(\mathbf{x})\right]$, so that $y(\mathbf{x})$ is $+1$ (resp. -1) when the number of perceptrons with output $+1$ (resp. -1) exceeds the number of perceptrons with output -1 (resp. $+1$), i.e., the output is decided by a voting among the perceptrons. In order to avoid ties [$\sum_{i=1}^n \text{sign}(\mathbf{w}_i^T\mathbf{x}) = 0$], an odd number n of perceptrons is required. In [2], the authors prove that the PPs behave as a universal approximator, and they propose the P-Delta rule to efficiently train a PP network. This rule uses ideas borrowed from statistical learning theory [9], updating the weights to get the right sign for the dot products $\mathbf{w}_i^T\mathbf{x}_k$ (for the N training patterns $\mathbf{x}_k, k = 1, \ldots, N$), but also to set a "security margin" $\gamma > 0$, which guarantees a good generalization ability (i.e., that noise or pattern variability in the test set will not carry $\mathbf{w}_i^T\mathbf{x}$ to the wrong sign). The P-Delta rule only updates $\mathbf{w}_i$ for the perceptrons i: 1) when the sign of $\mathbf{w}_i^T\mathbf{x}_k$ is not the right one for the desired output d_k, or 2) if this sign is the right one, but $\lvert\mathbf{w}_i^T\mathbf{x}_k\rvert < \gamma$ (meaning that a small variation in the dot product would lead it to the wrong sign).

The security margin γ is increased when all the perceptrons i verify $d_i\mathbf{w}_i^T\mathbf{x}_k > \gamma$, being reduced if some perceptron fails to meet this condition. Specifically, $\Delta\gamma = \eta(M_{\min} - \min\{M_{\max}, M_+ + M_-\})$, where $\eta = 1/4\sqrt{t}$ (being t the current training epoch), and M_+ (resp. M_-) is the number of perceptrons whose output has the right positive (resp. negative) sign but $0 \le \mathbf{w}_i^T\mathbf{x}_k < \gamma$ (resp. $-\gamma < \mathbf{w}_i^T\mathbf{x}_k \le 0$). The weight vectors must be normalized ($\lvert\mathbf{w}_i\rvert = 1$) in order to avoid that the weight norm influences the comparison between $\lvert\mathbf{w}_i^T\mathbf{x}_k\rvert$ and γ. The P-Delta rule is simultaneously local (because it takes into account the n outputs $\lvert\mathbf{w}_i^T\mathbf{x}_k\rvert$ of the single perceptron) and global (it also uses the network output y). Each weight is randomly initialized for each perceptron, so that the training evolves in parallel, but coordinated by the SWTA gate. In fact, since each perceptron can only learn linearly separable data, the parallel ensemble provides the ability to learn nonlinearly separable classification problems.

B. Analytical Calculation of Weights of a Single Perceptron

Now, let us consider a single perceptron in the PPs.[1] Taking into account the previous comments, the perceptron can be considered to have an error for the training pattern $\mathbf{x}_k$ when: 1) the desired output $d_k = +1$ but $\mathbf{w}^T\mathbf{x}_k < \gamma$, or 2) $d_k = -1$ but $\mathbf{w}^T\mathbf{x}_k > -\gamma$. This expression for the error can be written as a linear function of $\gamma - \mathbf{w}^T\mathbf{x}_k$ (for $d_k = +1$) and $\gamma + \mathbf{w}^T\mathbf{x}_k$ (for $d_k = -1$). Observing that the signs of d_k and $\mathbf{w}^T\mathbf{x}_k$ in the previous expressions are opposite, we can write the error for the training pattern $\mathbf{x}_k$ as $\gamma - d_k\mathbf{w}^T\mathbf{x}_k$ if $\gamma - d_k\mathbf{w}^T\mathbf{x}_k \ge 0$ and zero otherwise. Summing these errors over the training patterns $k = 1, \ldots, N$, the exact error (denoted by E^0) for a single perceptron can be defined as

$$E^0(\mathbf{w}) = \sum_{k=1}^{N}\left[\gamma - d_k\mathbf{w}^T\mathbf{x}_k\right]^+ \qquad [z]^+ = \max(z, 0). \tag{2}$$

This error measures two contributions. On one hand, the margin of the linear classifier is associated to the perceptron throughout the dot product $\mathbf{w}^T\mathbf{x}_k$. A low absolute value would allow that noise or pattern variability change the product sign, giving a wrong output in 1. This dot product is closely related to the distance $\lvert\mathbf{w}^T\mathbf{x}_k\rvert/\lvert\mathbf{w}\rvert$ from $\mathbf{x}_k$ to the hyperplane defined by $\mathbf{w}$, because $\lvert\mathbf{w}\rvert = 1$. In fact, the classification margin of a linear classifier is the minimum of these distances over all the training set,[2] which must be maximized in order to guarantee a good generalization ability. On the other hand, E^0 also measures the training error, because the product $d_k\mathbf{w}^T\mathbf{x}_k$ is positive only when d_k and $\mathbf{w}^T\mathbf{x}_k$ have the same sign [in this case, if the perceptron gives the right output for $\mathbf{x}_k$, the term $-d_k\mathbf{w}^T\mathbf{x}_k$ is negative and by (2) it reduces E^0]. Therefore, the error E^0 is simultaneously decreasing with the classification margin (related to its generalization ability, as the statistical learning theory shows) and increasing with the training (empirical) error, without the need for a regularization tunable parameter (usually denoted by C) used in the SVM to weigh both components.

The minimization of $E^0(\mathbf{w})$ would require $\nabla E^0(\mathbf{w}) = \mathbf{0}$, so that the function $[z]^+$ should be derivated. Also, the calculation of $\mathbf{w}$ which minimizes E^0 requires to invert the derivative of the function $[z]^+$. However, $\nabla E^0(\mathbf{w})$ cannot be calculated because $[z]^+$ is not derivable at $z = 0$. On the

[1]The input patterns $\mathbf{x}$ must have zero mean and standard deviation 1.

[2]In this paper, the error uses the sum of the dot products $d_k\mathbf{w}^T\mathbf{x}_k$ and not its minimum, because the minimum function is not differentiable.

other hand, the inverse of $[z]^+$ can only be defined for $z \geq 0$, because $[z]^+ \geq 0, \forall z$. Here, $z = \gamma - d_k \mathbf{w}^T \mathbf{x}_k < 0$ means that $d_k \mathbf{w}^T \mathbf{x}_k > \gamma$, so that the pattern $\mathbf{x}_k$ is correctly classified with high margin. In order to overcome these problems, we will use a linear approximation $[z]^+ \rightarrow z$: using the original $[z]^+$, the patterns with $d_k \mathbf{w}^T \mathbf{x}_k > \gamma$ do not contribute to the error, but using the linear approximation they reduce it. This negative contribution might compensate the positive contribution to the error of some other training patterns incorrectly classified ($d_k \mathbf{w}^T \mathbf{x}_k < 0$) or correctly classified with low margin ($0 < d_k \mathbf{w}^T \mathbf{x}_k < \gamma$). The validity of this linear approach will be tested in the experimental work. Now, the approximated error (denoted by E) can be written as

$$E(\mathbf{w}) = \sum_{k=1}^{N} \left(\gamma - d_k \mathbf{w}^T \mathbf{x}_k \right) = \gamma N - \sum_{k=1}^{N} d_k \mathbf{w}^T \mathbf{x}_k. \quad (3)$$

Since the derivative of E with respect to $\mathbf{w}$ does not depend on γ, this parameter does not influence on the weight vector $\mathbf{w}$ which minimizes E. Thus, using the linear approach we do not achieve a specific margin γ, but we simultaneously maximize the margin and the training accuracy. On the other hand, the optimal value of γ would be difficult to select in an eventual tuning process, which should be developed using the P-Delta rule. Thus, $\mathbf{w}$ will be calculated in order to have $|\mathbf{w}| = 1$ and to maximize $\sum_{k=1}^{N} d_k \mathbf{w}^T \mathbf{x}_k$, i.e., the training accuracy (positive terms in the sum) and the hyperplane margin (positive terms with high value in the sum). Taking into account that $\sum_k d_k \mathbf{w}^T \mathbf{x}_k = \mathbf{w}^T \sum_k d_k \mathbf{x}_k$, it is clear that the weight vector $\mathbf{w}_0$ with $|\mathbf{w}_0| = 1$ that minimizes E must be parallel to $\sum_k d_k \mathbf{x}_k$, so that it must be

$$\mathbf{w}_0 = \frac{\sum_{k=1}^{N} d_k \mathbf{x}_k}{\left| \sum_{k=1}^{N} d_k \mathbf{x}_k \right|}. \quad (4)$$

The (4) allows us to analytically calculate $\mathbf{w}_0$ using only the training patterns $\mathbf{x}_k$ and their desired outputs d_k. It is similar to the SVM, but without the N parameters α_i (the Lagrange multipliers), which verify $\alpha_i \neq 0$ only for the support vectors. Besides, these parameters must be calculated by solving a quadratic problem with a large computational cost, especially for high input dimensions d and numbers N of training patterns. Another difference between (4) and the SVM is that the weight vector of the SVM is on the hidden (or feature) space, and there is a kernel mapping from the input to the feature space, which can be (and usually is) nonlinear. Contrarily, (4) is like an SVM with all the training patterns as support vectors and $\alpha_i = 1,\ i = 1, \ldots, N$, so that the calculation of $\mathbf{w}_0$ is easy and very fast. On the other hand, and similar to the SVM with linear kernels, DPPs do *not* store the individual training patterns, but just the weight vector $\mathbf{w}_0$. However, the linear SVM still requires a complex training stage to calculate the Lagrange multipliers α_i and to select the support vectors. Another advantage is that DPPs can learn *online*, because a new training pattern $\mathbf{x}_{N+1}$ can be easily taken into account by adding it (multiplied by its desired output d_{N+1}) to $\mathbf{w}_0$ and normalizing $\mathbf{w}_0$ again. Again, this is not possible in linear SVMs, because each new pattern requires to retrain again on the whole training set.

Note that it may be $\sum_{k=1}^{N} d_k \mathbf{x}_k = \mathbf{0}$. In this case, by (3) we have $E = \gamma N$, which does not depend on $\mathbf{w}$. This special case is due to the linear approximation used for the error, and it would not happen using E^0, (2). The solution to this case will be explained for a network of PPs in the following subsection. On the other hand, taking into account that $x_{kd} = 1, k = 1, \ldots, N$ (the constant offset input), it follows that $w_{0d} = \sum_k d_k / |\sum_k d_k \mathbf{x}_k|$, and the sign of the offset w_{0d} depends only on the relative population of the two classes (remember that $d_k = \pm 1$). When both are equally populated, $\sum_{k=1}^{N} d_k = 0$ and $w_{0d} = 0$, so that the calculated hyperplane crosses the origin.

After calculating the weight value $\mathbf{w}_0$ that minimizes the error $E(\mathbf{w})$ subject to $|\mathbf{w}| = 1$, any weight vector $\mathbf{w}' = \mu \mathbf{w}_0, \mu > 0$ (i.e., parallel to $\mathbf{w}_0$) will give the same output $y(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$. Therefore, the normalization is not necessary for the output calculation, but it is required for the error minimization, because if $|\mathbf{w}|$ is not kept constant during this process, then $E(\mathbf{w}) = \gamma N - \mathbf{w}^T \sum_k d_k \mathbf{x}_k$ is linear in $\mathbf{w}$ without any minimum.

C. DPPs for Two-Class Problems

In a DPP network with n single perceptrons (with weights $\mathbf{w}_i$ and outputs $y_i, i = 1, \ldots, n$) and $N_c = 2$ classes, the total error E^0 is the sum of the errors defined in (2) for the n perceptrons. Using the linear approximation $[z]^+ \rightarrow z$, the approximate error E is given by

$$E = \sum_{i=1}^{n} \sum_{k=1}^{N} \left(\gamma - d_k \mathbf{w}_i^T \mathbf{x}_k \right) = \gamma nN - \sum_{i=1}^{n} \sum_{k=1}^{N} d_k \mathbf{w}_i^T \mathbf{x}_k. \quad (5)$$

Following similar calculations, we would come to the same result as in (4) and all the perceptrons would have the same weights, making the network unusable. An alternative is to use d_k to calculate a different desired output d_{ik} for each perceptron i. Since the output $y(\mathbf{x}_k)$ of the PP network is a voting among the perceptrons, each one can have a different desired output d_{ik}, provided their sum over the n perceptrons has the same sign as d_k. The value d_{ik} can be randomly generated with values in the set $\{\pm 1\}$

$$d_{ik} = \text{rand}(\{-1, 1\}), i = 1, \ldots, n: \qquad d_k \sum_{i=1}^{n} d_{ik} > 0. \quad (6)$$

Specifically, we initially set $d_{ik} = d_k, i = 1, \ldots, n$, then, we generate $m = (n-1)/2$ (m is an integer because n is always odd, as commented in Section III-A) random numbers $\{\xi_p\}_{p=1}^{m}, \xi_p \in \{1, \ldots, n\}$ and set $d_{\xi_p k} = -d_{\xi_p k}, p = 1, \ldots, m$. Since $m < n/2$, the number of perceptrons in PPs with $d_{ik} = d_k$ is always higher than the number of perceptrons with $d_{ik} = -d_k$. The values $\{d_{ik}\}_{i=1}^{n}$ are used as desired outputs for the n perceptrons and the training pattern $\mathbf{x}_k$. Thus, the error $E(\mathbf{w}_1, \ldots, \mathbf{w}_n)$ and the weight vectors $\mathbf{w}_{i0}, i = 1, \ldots, n$,

which minimize E are given by

$$E = \gamma nN - \sum_{i=1}^{n}\sum_{k=1}^{N} d_{ik}\mathbf{w}_i^T\mathbf{x}_k;\ \mathbf{w}_{i0} \equiv \frac{\sum_{k=1}^{N} d_{ik}\mathbf{x}_k}{\left|\sum_{k=1}^{N} d_{ik}\mathbf{x}_k\right|},\quad i = 1,\ldots,n. \tag{7}$$

Each single perceptron i, $i = 1,\ldots,n$, "sees" a different training set $\{\mathbf{x}_k, d_{ik}\}_{k=1}^{N}$, which has a set of desired outputs different to the other perceptrons. This randomization of d_{ik} for each perceptron in DPPs is analogous to the weight initialization in the P-Delta rule and other gradient descent training approaches, because it fixes several "different views" of the problem for each perceptron. In the P-Delta rule, the different weight initializations are the starting points for an iterative training process, which does not exist in DPPs, but in both cases they allow calculation of different weight values for each perceptron. In the practice, we train several random initializations of the perceptron's desired outputs d_{ik} in DPPs, and we select the one that provides the highest accuracy on the training set. It may be argued that the randomization of the desired outputs makes DPPs an "iterative procedure," and not a "direct" method. However, this randomization repeats the weight calculation a prespecified number of times, and it does not require a convergence process which may be faster or slower, and it might oscillate or not to converge at all as in the P-Delta rule or backpropagation. Besides, a high number of randomizations is not necessary, as we will show in Section V. Finally, if the dataset $\{\mathbf{x}_k, d_{ik}\}_{k=1}^{N}$ for the perceptron i verifies $\sum_{k=1}^{N} d_{ik}\mathbf{x}_k = \mathbf{0}$, as we noted in the previous subsection, the output will be the same for all the training patterns $\mathbf{x}_k$, and the accuracy on the training set will be low. Consequently, another initialization $\{d'_{ik}\}$ verifying $\sum_{k=1}^{N} d'_{ik}\mathbf{x}_k \neq \mathbf{0}$, which will provide better accuracy on the training set, will be selected for the weight calculation.

D. Complete Algorithm—Two-Class Problems

The whole procedure[3] for the DPPs and $N_c = 2$ classes is the following.

1) **Inputs**: $\{\mathbf{x}_k = (x_{k1},\ldots,x_{k(d-1)},1), d_k\}_{k=1}^{N}$ are the training patterns and desired outputs, respectively. The constant bias or offset input is $x_{kd} = 1$ for all the training and test patterns. Let be n the (odd) number of perceptrons.
2) **Preprocessing**: $x_{kj} = (x_{kj} - \mu_j)/\sigma_j, k = 1,\ldots,N, j = 1,\ldots,d-1$, μ_j and σ_j bring the mean and standard deviation of input j, calculated using the training patterns. Note that the input d (which is constant and equal to 1) is *not* preprocessed.
3) **Calculation of the desired outputs d_{ik} for each perceptron**: Let $m = (n-1)/2$ (n is an odd number). For each training pattern $\mathbf{x}_k, k = 1,\ldots,N$, let $d_{ik} = d_k, i = 1,\ldots,n$, generate m integer random numbers $\{\xi_p\}_{p=1}^{m}$ with $\xi_p \in \{1,\ldots,n\}$. Let $d_{\xi_p k} = -d_{\xi_p k}$, $p = 1,\ldots,m$.
4) **Calculation of the weights** using (7) (right)

$$\mathbf{w}_{i0} = \frac{\sum_{k=1}^{N} d_{ik}\mathbf{x}_k}{\left|\sum_{k=1}^{N} d_{ik}\mathbf{x}_k\right|},\qquad i = 1,\ldots,n.$$

5) **Test**: After preprocessing the input pattern $\mathbf{x}$ using $x_j = (x_j - \mu_j)/\sigma_j$, $j = 1,\ldots,d$, the output of the DPPs is given by (8)

$$y(\mathbf{x}) = \operatorname{sign}\left[\sum_{i=1}^{n} \operatorname{sign}\left(\mathbf{w}_{i0}^T\mathbf{x}\right)\right]. \tag{8}$$

E. DPP for Multiclass Problems

The PPs in [2] are designed for two-class problems, because the network output $y(\mathbf{x}) \in \{\pm 1\}$. With $N_c > 2$ classes, the "one-against-all" (OAA) approach, widely used with SVM, can be applied.[4] We need N_c DPPs $l = 1,\ldots,N_c$ (let n_l be the number of perceptrons of the DPPs l, let $\mathbf{w}_{li}$ and y_{li} be the weight vector and output, respectively of the perceptron i in the DPPs l), and the DPPs l will discriminate between the patterns belonging to the class l and the remaining ones. Given the desired output $d_k \in \{1,\ldots,N_c\}$ for the training pattern $\mathbf{x}_k$, the desired output d_{lk} for the DPPs l is defined as

$$d_{lk} = \begin{cases} +1, & d_k = l \\ -1, & d_k \neq l. \end{cases} \tag{9}$$

Using d_{lk}, the desired ouputs d_{lik} and the weight vectors $\mathbf{w}_{li0}, i = 1,\ldots,n_l$ are calculated similar as in the previous subsection [(6) and (7), respectively]

$$d_{lik} = \operatorname{rand}(\{-1,1\}):\quad d_{lk}\sum_{i=1}^{n_l} d_{lik} > 0 \tag{10}$$

$$\mathbf{w}_{li0} \equiv \frac{\sum_{k=1}^{N} d_{lik}\mathbf{x}_k}{\left|\sum_{k=1}^{N} d_{lik}\mathbf{x}_k\right|},\ l = 1,\ldots,N_c,\quad i = 1,\ldots,n. \tag{11}$$

During the test stage, the input pattern $\mathbf{x}$ is preprocessed using $x_j = (x_j - \mu_j)/\sigma_j$, $j = 1,\ldots,d-1$, and then it is assigned to the class l for which $y_l(\mathbf{x}) = 1$, where $y_l(\mathbf{x})$ is the output of the DPPs l, given by

$$y_l(\mathbf{x}) = \operatorname{sign}\left[\sum_{i=1}^{n_l} \operatorname{sign}\left(\mathbf{w}_{li0}^T\mathbf{x}\right)\right]. \tag{12}$$

It may happen that several DPPs (associated with different classes) give output $+1$ (there would be a tie) or that all the DPPs give output -1 (then the pattern would not be assigned to any class). The situation is also possible in multiclass SVM (using the OAA approach) or MLP, and in both cases the pattern is assigned to the perceptron with the highest output.

[3] Available at: http://www.gsi.dec.usc.es/~delgado/programs/dpp_two_class.c.

[4] Available at: http://www.gsi.dec.usc.es/~delgado/programs/dpp_multi_class.c.

TABLE II

DATA SETS: NAME, KEY USED IN THE TABLES, NUMBER OF PATTERNS, INPUTS AND CLASSES, PERCENTAGE OF PATTERNS OF THE MAJORITY CLASS (%MJ), AND PERCENTAGE IF CLASSES WERE EQUALY PROBABLE (%EQ = 100/#CLASSES). *SUM OF TRAINING AND TEST PATTERNS

Name	Key	#Patterns	#inputs	#class	%MJ	%EQ
Sonar	SNR	208	60	2	53.4	50
Heart	HRT	297	13	2	54.1	50
Ionosphere	ION	351	34	2	64.1	50
Breast	BST	683	9	2	65.5	50
Pima	PIM	768	8	2	65.1	50
German	GRM	1000	24	2	70	50
KRKPA7	KP7	3196	36	2	52.2	50
Thyroid	TYR	3772	21	2	92	50
Zoo	ZOO	101	16	7	40.5	14.3
Wine	WIN	178	13	3	39.9	33.3
Glass	GLA	214	9	7	35.5	14.3
Contrac.	CON	1473	9	3	42.7	33.3
Yeast	YST	1484	8	10	31.2	10
Abalone	ABA	4177	8	3	34.7	33.3
Handwritten	HND	5620*	64	10	10.2	10
USPS	USP	9298*	256	10	17.1	10
Nursery	NUR	12960	8	5	33.3	20
Letter	LET	20000	16	26	4.1	3.8
Shuttle	SHU	58000	9	7	78.6	14.3
MNIST	MNI	70000*	780	10	11.3	10

In these cases, the pattern $\mathbf{x}$ is assigned to the class l for which $\sum_{i=1}^{n_l} \mathbf{w}_{li0}^T \mathbf{x}$ is the highest

$$y(\mathbf{x}) = m \in \{1, \ldots, N_c\} : \begin{cases} y_m(\mathbf{x}) = 1 & y_l(\mathbf{x}) \neq 1 \quad \forall l \neq m \\ m = \arg\max \left\{ \sum_{i=1}^{n_l} \mathbf{w}_{li0}^T \mathbf{x} \right\}_{l=1}^{N_c} & \text{otherwise}. \end{cases} \tag{13}$$

When more than one DPP have the same output $\sum_{i=1}^{n_l} \mathbf{w}_{li0}^T \mathbf{x}$, there is a tie among these classes and the multiclass PPs would not be able to classify the test pattern $\mathbf{x}$, so the output would be the class with the smallest index l. Again, this might also happen in the MLP or multiclass SVM when the same (maximum) value is given by several outputs.

IV. RESULTS

In this section, we will report the results on real two-class and multiclass benchmark datasets.

A. Two-Class Datasets

We compared DPPs with the PPs trained using the P-Delta rule on the same two-class datasets used in [2] (see Table II), selected from the UCI machine learning database:[5] *Sonar, Mines versus Rocks* (briefly Sonar, SNR), *Heart Disease* (Heart, HRT), Ionosphere (ION), *Breast Cancer Wisconsin* (Breast, BST), *Pima Indian Diabetes* (Pima, PIM), *German Credit Data* (German, GRM), *Chess End-Game – King+Rook versus King+Pawn on a7* (KRKPA7, KP7) and *Thyroid Disease* (called Sick in [2], but we will call it Thyroid, TYR).

[5]Available at: http://archive.ics.uci.edu/ml.

TABLE III

RESULTS ON TWO-CLASS DATASETS: ACCURACIES IN % AND FRIEDMAN RANKING (DECREASING WITH THE ACCURACY) OF EACH APPROACH. THE BEST VALUES ARE IN BOLD. *RESULTS REPORTED FROM [2]

Set	DPPs ($n = 3$)	PD* ($n = 3$)	MD*	MLP* ($n = 3$)	C4.5*
SNR	78.6	74.0	78.8	81.6	73.3
HRT	**90.0**	80.0	78.8	82.1	76.2
ION	87.5	84.8	86.5	89.4	89.7
BST	**97.4**	96.9	92.3	96.5	95.5
PIM	77.5	73.7	73.4	76.8	73.7
GRM	**79.7**	71.7	70.5	73.1	72.7
KP7	87.2	97.2	98.0	99.3	**99.4**
TYR	98.8	95.7	95.7	96.2	98.7
Avg.	87.1	84.3	84.3	86.9	84.9
Rank.	4.12	7.56	8.37	5.31	6.68

Set	P-SVM*	L-SVM	AB-5MLPs	BG-5MLPs	LDA
SNR	84.5	74.8	99.0	**99.1**	73.0
HRT	80.8	87.3	83.8	82.81	84.5
ION	**91.2**	88.9	89.8	89.7	90.3
BST	96.9	97.1	96.8	95.9	94.5
PIM	77.3	**77.7**	76.7	73.2	75.9
GRM	75.4	75.0	74.6	73.0	76.3
KP7	**99.4**	95.8	99.3	99.2	93.5
TYR	93.9	98.9	**99.5**	99.3	98.5
Avg.	87.4	86.9	**89.9**	89.0	85.8
Rank.	4.06	4.25	**3.56**	5.18	5.87

The patterns with missing values were removed in Breast. The last two columns in Table II report the percentage of patterns belonging to the majority class (column %MJ) and the same percentage if all the classes would have the same number of patterns (column %EQ) the classes are equally populated when the two values are similar.

For all the approaches in the experimental work, we used the same methodology as in [2] 10-fold cross validation (90% of the patterns for training and the remaining 10% for testing) in order to compare the different approaches. With DPPs, we observed that the value of n does not influence very much the results, so we used $n = 3$ to compare with P-Delta in [2]. We selected the best accuracy over 50 randomizations of the desired outputs d_{ik}, although a lower number (e.g., 10) could be used. In fact, we achieved similar results with numbers of randomizations between 10 and 100. Table III shows the accuracy achieved by the different approaches. The values achieved by **P-Delta** with $n = 3$, **Madaline** [20], **MLP** (three hidden neurons with sigmoid activation trained using backpropagation), **C4.5**, and **P-SVM** (SVM with second degree polynomial kernel) are reported from [2] using the WEKA implementation [21]. Additionally, linear kernel SVM (**L-SVM** in Table III), using LibSVM (version 2.84),[6] was compared with DPPs. The regularization parameter C of L-SVM was tuned using values in the range $\{2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{14}\}$ (20 values). We also tried two machine committee algorithms, **Adaboost** (AB) [22] and **Bagging** (BG) [23] of MLP networks, using

[6]Available at: http://www.csie.ntu.edu.tw/cjlin/libsvm.

TABLE IV
SELECTED VALUES FOR THE REGULARIZATION PARAMETER C (G-SVM AND L-SVM) AND SPREAD σ (G-SVM) ON MULTICLASS DATASETS

SVM		ZOO	WIN	GLA	CON	YST	ABA
G	C	16384	16384	16384	16384	16384	16384
	σ	2.4E-4	3E-5	6.25E-2	6.25E-2	1.0	0.5
L	C	0.25	0.5	8192	0.0625	16384	256
		HND	USPS	NUR	LET	SHU	MNIST
G	C	4	4	16384	64	1	16384
	σ	1.96E-3	1.96E-3	0.5	0.25	1	1.95E-3
L	C	1.5E-2	1.96E-3	0.5	0.25	1	1.95E-3

the Torch [24][7] implementation. BG and AB used five MLPs with hidden neurons in the range $\{3, 5, 7, 10, 15\}$ (this parameter was tuned on the training set) and the default Torch parameters (we used the class negative log-likelihood criterion, recommended in Torch for classification problems, with 100 training epochs, learning rate = 0.01 and error goal = 1E–5), although we developed experiments changing these values with similar or worse results. LDA [25] was also tested, using the R (version 2.10.1)[8] implementation. We also calculated the average accuracy over all the datasets, and developed the Friedman test [26], using a publicly available tool,[9] to rank the different approaches (last two rows in Table III).

B. Multiclass Datasets

We also developed experiments using multiclass datasets (Table II), including *Zoo* (ZOO), *Wine* (WIN), *Glass* (GLA), *Contraceptive method* (CMC), *Yeast* (YST), *Abalone* (ABA), *Handwritten data digits* (HND), *USPS*[10] (USP), *Nursery* (NUR), *Letter Recognition* (LET), Statlog *Shuttle* (SHU), and *MNIST*[11] (MNI). We used the multiclass DPPs described in Section III-E with $n_l = 3, l = 1, \ldots, N_c$ (3 perceptrons for each DPPs) and the same configuration as in the previous subsection. In order to compare DPPs and P-Delta, we developed our own implementation of the P-Delta rule for multiclass datasets, using a PP network (with $n = 3$ perceptrons) for each class and the OAA approach, similarly to DPPs. The multiclass P-Delta rule assigns the input pattern to the class associated to the PPs with the highest net sum $\sum_{i=1}^{N} \mathbf{w}_i^T \mathbf{x}$. As far as we know, this is the first time P-Delta is applied to multiclass problems. P-Delta used the recommended values for the parameters: $\gamma = 0.05$, $\epsilon = \rho = 0$, $\mu = 1$, $\eta = 0.25$, $\beta_{\min} = 0.05$, and $\beta_{\max} = 0.2$. The training followed the same early stopping criterion of [2, p. 792], but for most of the datasets 2000 training epochs were required to achieve accuracies similar to Table III. Besides, P-Delta used 10 weight randomizations in order to achieve these accuracies.

We also tested BG and AB of five MLPs, with the same settings as in the previous subsection, and SVM, using LibSVM with Gaussian and linear kernels (denoted as G-SVM and L-SVM, respectively). We did not use polynomial kernels because they have two tunable parameters (offset and degree), while the Gaussian kernels have only one (spread) and achieved better results in our experiments. The regularization parameter C was tuned using the same values as L-SVM in two-class datasets, and the spread σ in G-SVM used values in the range $\{2^{-15}, 2^{-14}, 2^{-13}, \ldots, 2^{2}\}$ (18 values). Both G-SVM and L-SVM used the same 10-fold cross-validation methodology, tuning the parameters over the training set and selecting the values reported in Table IV. LibSVM uses the "one-against-one" [27] approach for multiclass classification problems, as opposed to DPPs and P-Delta (however, we developed experiments using G-SVM with the OAA approach achieving similar results). In the dataset MNIST, L-SVM did not converge using LibSVM, and we used LibLinear (version 1.7),[12] recommended by the LibSVM creators for large-scale datasets.

TABLE V
RESULTS ON MULTICLASS DATASETS: ACCURACIES AND FRIEDMAN TEST RANKING FOR EACH APPROACH. THE BEST VALUES ARE IN BOLD

Set	DPPs ($n = 3$)	PD ($n = 3$)	G-SVM	L-SVM
ZOO	96.4	95.5	92.7	96.4
WIN	96.4	94.4	92.2	97.8
GLA	69.1	54.1	**73.2**	64.5
CON	51.3	50.9	44.3	52.3
YST	55.3	38.9	56.8	60.7
ABA	55.2	50.3	**67.0**	64.7
HND	85.1	33.5	**97.4**	96.8
USP	76.5	73.5	**95.1**	93.4
NUR	87.2	48.30	**99.9**	90.6
LET	43.4	54.1	**98.0**	84.7
SHU	86.7	68.9	**99.9**	98.9
MNI	71.3	75.8	93.6	67.8
Avg.	72.8	60.1	84.2	80.7
Rank.	6.04	7.33	3.71	4.33

Set	AB-5MLPs	BG-5MLPs	LDA	Other
ZOO	**98.0**	96.0	72.9	ECN: 95.6
WIN	98.8	98.4	96.6	QDA: **99.4**
GLA	72.7	68.8	57.5	SVMB: 61.9
CON	**55.9**	55.1	51.0	ECN: 55.2
YST	59.76	**61.0**	60.72	ECN: 59.6
ABA	66.5	66.7	62.9	MLP: 65.6
HND	96.3	96.7	93.4	KNN: 98.0
USP	92.1	92.3	88.3	TD: 97.5
NUR	**99.9**	**99.9**	38.2	ECN: 90.7
LET	88.35	88.1	70.0	KNN: 93.6
SHU	**99.9**	**99.9**	94.41	KNN: **99.9**
MNI	94.5	94.6	87.3	CNN: **99.6**
Avg.	**85.2**	84.8	72.8	84.7
Rank.	**2.83**	2.95	5.95	**2.83**

Table V shows the accuracy of DPPs, P-Delta (PD, $n = 3$), G-SVM, L-SVM, AB-5MLPs, BG BG-5MLPs), and LDA.

[7] Available at: http://www.torch.ch.

[8] Available at: http://www.r-project.org.

[9] Available at: http://sci2s.ugr.es/sicidm/eight.

[10] Available at: http://www-i6.informatik.rwth-aachen.de/keysers/usps.html.

[11] Available at: http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/multiclass.html#mnist.

[12] Available at: http://www.csie.ntu.edu.tw/cjlin/liblinear.

TABLE VI
EXECUTION TIMES (IN SECONDS) OF DPPS, P-DELTA, G-SVM, AND L-SVM. *P-DELTA WAS RUN WITHOUT EARLY STOPPING (2000 EPOCHS)

Two-classes	DPPs	P-Delta	G-SVM	L-SVM
SNR	0.713	6.634	56.0	14.36
HRT	0.175	244.93*	52.15	2882
ION	0.966	518.91*	71.07	4699
BST	0.302	1.45	68.79	237
PIM	0.660	2.66	604	3801
GRM	0.978	6.37	856.3	22164
KP7	5.249	22.97	26275	24615
TYR	4.100	4589*	1072	1980
Avg.	1.64	674	3632	7549
Multiclass	**DPPs**	**P-Delta**	**G-SVM**	**L-SVM**
ZOO	0.579	4.619	6.24	0.852
WIN	0.383	5.114	29.79	0.876
GLA	0.728	4.310	28.47	170.2
CON	2.716	12.753	2423	9940
YST	5.614	102.27	1517	34721
ABA	5.462	23.97	10108	29067
HND	8.378	4247.8*	6954.6	67.2
USP	93.01	30294*	80035	761
NUR	94.28	4736*	9022	7835
LET	709.72	21026*	213922	604414
SHU	896.53	96797*	209728	165395
MNI	1178.1	68828	25693700	825600
Avg.	249.6	18840	2185623	94500

The last column ("Other") reports the accuracy of other approaches in the literature on the same datasets (in some cases, 10-fold cross validation was not used): ECN = error counting network [28], QDA = quadratic discriminant a [15], SVMB = BG of SVMs [29], MLP [15], KNN = K-nearest neighbors (for handwritten [15], Letter[13] and Shuttle [30]), TD = tangent distance [31], CNN = convolutional neural network [32]. The average over all the datasets and the Friedman test ranking are in the two last rows. Given that the datasets Handwritten, USPS, and MNIST have separate training and test sets (and all the references in the literature report results on the test sets), all the classifiers used them for training and testing, respectively, instead of the 10-fold cross validation used for the other datasets. Some datasets (Abalone, Yeast, Zoo, Letter, and Nursery) had discrete attributes, whose values were codified as equally spaced numbers in the range $[-1, 1]$. There was no dataset with missing values.

C. Execution Times

Table VI compares the elapsed times (training, parameter tuning and test, 10-fold cross validation) required by DPPs, P-Delta, G-SVM, and L-SVM for two-class and multiclass datasets. The trials were developed with C and C++ programs compiled with gcc/g++ 4.2.1 (with the −g option) under Linux Debian 4 (Etch) 2.6.18-6-686 executed on an AMD Athlon 64 Processor 3000+ with 1.48 GB RAM. We used the same experimental setting of the previous subsections. In the datasets labeled *, early stopping was disabled for P-Delta in order to achieve better accuracy. Fig. 1 shows the time (in seconds and logarithmic scale) spent by the four approaches for two-class (upper left panel) and multiclass (upper right panel) datasets. The lower left panel shows the times of the three approaches divided by the number of patterns for two-class datasets, plotted against the number d of inputs. The lower right panel shows the times (divided by the number of patterns and classes) plotted against the number of inputs for multiclass datasets.

[13]Available at: http://www.is.umk.pl/projects/datasets-stat.html#Letters.

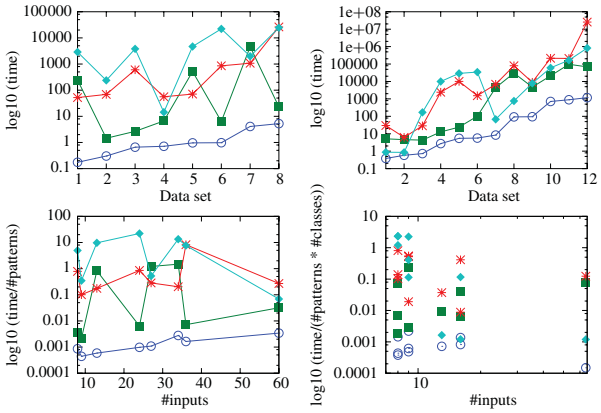

Fig. 1. Logarithmic plots of the elapsed times (in seconds) spent by DPPs (circles), P-Delta (squares), G-SVM (asterisks), and L-SVM (diamonds). Upper left panel: time spent by the four approaches on each two-class dataset (ordered by increased times of DPPs). Upper right panel: the same plot for multiclass datasets. Lower left panel: times divided by the number of patterns against the number of inputs for two-class datasets. Lower right panel: times divided by the number of patterns and classes against the number of inputs for multiclass datasets.

V. DISCUSSION

The results will be discussed in the following subsections for two-class and multiclass datasets for the dataset properties, comparison with P-Delta, and in terms of execution times.

A. Two-Class Datasets

In two-class problems (Table III), DPPs are the third best approach, after AB and P-SVM, achieving the highest accuracy in three of eight datasets (Heart, Breast, and German). Besides, its accuracy is very similar to the best approach in datasets Pima (77.5% against 77.7% using L-SVM), Ionosphere (87.5% against 91.2% using P-SVM), and Thyroid (98.8% against 99.5% using AB). In Sonar, DPPs achieve 78.6% (similarly to P-Delta, Madaline, and C4.5) against 99.1% using AB and BG, and in KRKPA7 DPPs achieve 86.1% against 99.9% using the others. On average (although the ten approaches are in a narrow range between 84%–90% of accuracy), AB achieves the highest accuracy (89.9%), followed by BG (89%), P-SVM (87.4%), and DPPs (87.1%), only 0.5 points below SVM and slightly better than MLP (86.9%). Finally, LDA, C4.5, P-Delta, and Madaline are the worst ones (about 84%). The Friedman test ranking (line "Rank" in Table III) shows that AB is the best approach (3.56), followed at a certain distance by P-SVM (4.06) and DPPs (4.12). BG (5.18) is below DPPs in the ranking due to its bad results on

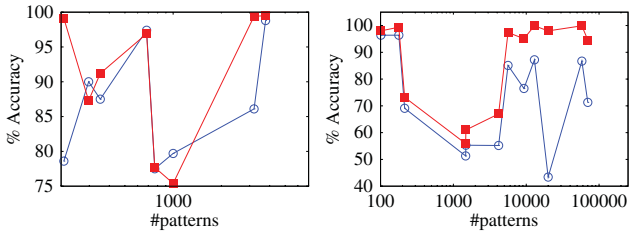

Fig. 2. Accuracy achieved by DPPs (circles) and the best classifier (squares) plotted against the number of patterns for two-class (left panel) and multiclass datasets (right panel).

TABLE VII
AVERAGE DIFFERENCE $\Delta E = 100|E - E^0|/E^0$ BETWEEN E^0 AND E [REAL AND APPROXIMATED ERROR, RESPECTIVELY SEE (2) AND (3)]

Data	SNR	HRT	ION	BST	PIM	GRM	KRKPA7
Av.Dif.	14.2	19.9	31.4	10.5	34.4	22.9	14.8
Data	TYR	ZOO	WIN	GLA	CON	YST	ABA
Av.Dif.	1.9	77.6	8.8	20.7	45.8	48.0	36.5
Data	HND	USP	LET	NUR	SHU	MNI	
Av.Dif.	47.3	54.6	103.3	4.4	4.6	27.2	

datasets Breast, Pima, and German compared to DPPs. MLP (ranking 5.31) is worse than BG, and the worst approaches are LDA (5.87), C4.5 (6.68), P-Delta (7.56), and Madaline (8.37). Therefore, DPPs achieve on two-class datasets similar accuracy as P-SVM (both are below AB), it is better than BG and MLP, and clearly better than LDA, C4.5, P-Delta, and Madaline. In fact, if KRKPA7 and Sonar datasets were not considered, DPPs would achieve the highest average accuracy (88.5%) and the best Friedman ranking (2.83), followed by AB (86.9% accuracy, ranking 3.67), P-SVM, L-SVM, and LDA (4.5), MLP (5.67), and the other approaches.

B. Multiclass Datasets

In multiclass datasets (Table V), the performance of DPPs is near to the reference values in 5 of 12 datasets: Zoo, Wine, Glass, Contraceptive, and Yeast (in this case, the difference with respect to the best classifier is about 5%). However, DPPs are below the best performance (G-SVM) in MNIST and USPS (by 20%), and in Abalone, Handwritten, Nursery, and Shuttle (by about 12%). In the Letter dataset DPPs achieve 43.4% accuracy, which shows that they are not able to correctly learn this dataset. Nevertheless, the average accuracy achieved by DPPs (71.3%) is 10 points above P-Delta (60.1%), and DPPs are always more accurate than P-Delta except for the Letter dataset. The average accuracy and the Friedman test ranking for the multiclass datasets show clearly that the best approach is AB (85.2%, ranking 2.83), followed by BG (84.8%, ranking 2.95), G-SVM (84.2%, ranking 3.71), and L-SVM (76.0%, ranking 4.33), LDA and DPPs being similar (72.8%, rankings 5.95 and 6.04, respectively) and P-Delta the worse approach (60.1%, ranking 7.33).

C. Dependence on Data Properties

There is no clear relation between the results achieved by DPPs and the number of inputs or the relative population of the classes. In twoclass datasets, DPPs achieve the worst results on KRKPA7, with 36 inputs and %MJ near 50% (Table II, the classes are equally populated), and it achieves the best results on Thyroid, with 21 inputs and %MJ equal to 92% (the classes are not equally populated). So, the accuracy of DPPs is also satisfactory for some datasets whose classes are not equally populated. In multiclass datasets, the results of DPPs are acceptable in some datasets with high %MJ (e.g., in Zoo, where %MJ is 40.5% and %EQ is 14.3%, i.e., one of seven classes has the 40.5% of the patterns, or Glass, where %MJ is 35.5% and %EQ is 14.3%), while the results are suboptimal on datasets Abalone, where the classes are equally populated (the most populated class has 34.7% of the patterns, and equally populated classes should have 33.3%) and Letter (4.1% against 3.8%). The performance of DPPs does not reduce with the number of classes, because it achieves good results for Glass and Yeast (7 and 10 classes, respectively) and worse results for Abalone (3 classes). The number of patterns does not influence the accuracy for two-class datasets in Fig. 2 (left panel), the difference between DPPs and the best classifier (when DPPs are the best, then the second one is plotted) does not increase with the number of patterns. However, in the multiclass datasets (right panel) the difference between DPPs and the best classifier seems to be higher for highly populated datasets.

It is interesting to verify the quality of the approximation done by replacing the error E^0 by E [(2) and (3), respectively]. Table VII reports the average difference between E^0 and E, as a percentage of E^0, defined as $\Delta E \equiv 100|E - E^0|/E^0$, using $\gamma = 0.05$ (recommended value for γ in [2]). The values are very variable among datasets, and in some cases they are high (up to 103.3% in Letter, 77.6% in Zoo, 54.6% in USPS, 48.0% in Yeast), being lower for the other datasets. It seems that the worst results of DPPs are for datasets with high differences (Letter, USPS, and Handwritten), although there are exceptions, e.g., the dataset Zoo, also with high diference (77.6%) but relatively good results (DPPs achieve 96.4% against 98% using AB-5MLPs). Besides, for many datasets the difference between E^0 and E is high but the accuracy of DPPs is also high (e.g., Ionosphere, Zoo, Contraceptive, and Yeast), which means that DPPs may work well even when E is not a good approach to E^0.

D. Kernelization and Comparison with P-Delta

We also explored other approaches, alternatives to a network of PPs, for the classification of nonlinearly separable datasets, but using the same expression to calculate the weights (4). Specifically, we used just one perceptron, with weight vector calculated directly as in DPPs but a Gaussian kernel $K(\mathbf{x}, \mathbf{y}) = e^{-|\mathbf{x}-\mathbf{y}|^2/\sigma^2}$ to learn nonlinearly classification borders in two-class problems [33]. Considering that now the training patterns $\mathbf{x}_k$ in (4) are in the feature space, they must be replaced by $\Phi(\mathbf{x}_k)$, being Φ the nonlinear mapping, and using $\Phi(\mathbf{x}_k)^T \Phi(\mathbf{x}) = K(\mathbf{x}_k, \mathbf{x})$ we achieve the following expression

for the classifier output:

$$y(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{N} d_k K(\mathbf{x}_k, \mathbf{x})\right]. \quad (14)$$

We tested this approach on the multiclass datasets where DPPs achieved low performance, using also the OAA approach, and we achieved accuracies better than DPPs: 96.4% in Handwritten, 93.1 in USPS, 93.8% on Nursery, 94.0% in Letter, 99.8% in Shuttle, and 74.4% in MNIST. These results suggest that the low efficiency of DPPs for some datasets is not caused by the direct weight calculation (4), because the same expression applied on the feature space provides competitive results. Indeed, it seems that the parallel combination of single perceptrons in [2], despite of being universal approximators with good performance and biological plausibility, may not be so competitive as SVM and other tools for large-scale problems. In fact, the results of P-Delta are also suboptimal on these large-scale datasets.

Additionally, we tested DPPs and P-Delta rule on the well-known "Two-spirals-apart" dataset, which is also nonlinearly separable. We used 200 patterns, 90% for training and 10% for testing. Fig. 3 shows (upper left panel) the borders learnt by P-Delta using $n = 75$ perceptrons, and it confirms that P-Delta was not able to learn this dataset even with many perceptrons. This also happens with DPPs (upper right panel) with the same experimental settings. We also tested P-Delta by varying the number n of perceptrons in the set $\{3, 5, 9, 13, 15, 19, 25, 29, 35, 51, 75\}$ (lower left panel), but the accuracy never crossed 65% (the base is 50% since both classes are equally populated). On the contrary, a direct perceptron, using (14) with a Gaussian kernel and spread $\sigma = 0.0625$ achieved 95.8% accuracy in learning the borders, as shown in the lower right panel. This results also suggest that kernelization of a single perceptron trained using (14) seems more competitive than a parallel combination of single perceptrons using DPPs or P-Delta.

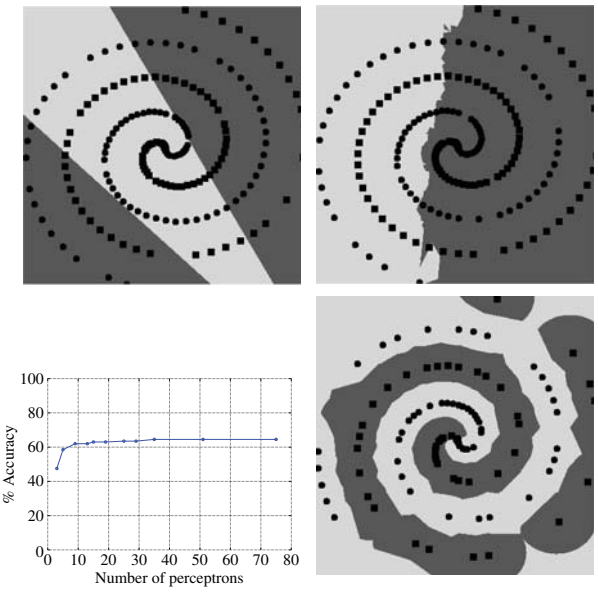

Fig. 3. Results of P-Delta and DPPs on the two-spirals-apart dataset. Upper left panel: classification borders for P-Delta with $n = 75$ perceptrons (200 patterns, 90% for training and 10% for testing). Upper right panel: the same borders for DPPs using the same experimental setting. Lower left panel: Classification accuracy of P-Delta against the number n of perceptrons. Lower right panel: borders learnt by only one direct perceptron [with weights calculated using (4)] with Gaussian kernel using spread $\sigma = 0.0625$, achieving 95.8% accuracy on the same dataset.

E. Execution Times

Table VI and Fig. 1 shows, that the elapsed times of DPPs are dramatically lower than P-Delta, G-SVM, and L-SVM both for two-class (1.64 s on average against 674 s for P-Delta, 3632 s for G-SVM, and 7549 s for L-SVM) and multiclass datasets (249.63 s against 18 840 s for P-Delta, 2 185 623 s for G-SVM, and 94 500 s for L-SVM). On average, P-Delta is 386 times slower than DPPs, G-SVM is 967 times, and L-SVM is 6967 times for two-class datasets. For multiclass datasets, P-Delta is 94 times slower, G-SVM 2272 times, and L-SVM 1372 times. P-Delta was faster than G-SVM for the two-class datasets, except in Heart, Ionosphere, and Thyroid, where early stopping was not used. On multiclass datasets, P-Delta was always faster than G-SVM (on average 86 times faster). The times of L-SVM are higher than G-SVM for some datasets, especially with two classes, despite having just one tunable parameter (C), while G-SVM has two (C and the Gaussian kernel spread σ). The reason might be the nonlinear separability of some datasets, which would slow down the training using linear kernels. These times are very variable, L-SVM being the slowest for datasets 3–6 in Fig. 1, upper right panel, faster than P-Delta for sets 1, 2, 7, 8, and between G-SVM and P-Delta in the large datasets (9–12). On average, L-SVM is slower than G-SVM (7549 s against 3632 s) for two-class datasets, and faster than G-SVM (95 400 s against 2 185 623 s) for multiclass datasets. In fact, by dividing by the number of values of the tunable parameters (20 values of C for L-SVM, 20 values of $C \times 18$ values of $\sigma = 360$ values for G-SVM) we achieve 10.1 s for G-SVM and 3477.4 s for L-SVM in two-class datasets. For multiclass datasets, these values are 6071 s for G-SVM and 5250 s for L-SVM.

Fig. 1 shows in the **upper left panel** that the times of P-Delta, G-SVM, and L-SVM rise much more quickly than DPPs, and usually G-SVM and L-SVM are slower than P-Delta, which depends very much on the early stopping. The graph of L-SVM is usually above G-SVM, but also with strong variations. The **upper right panel** also shows that DPPs are clearly faster than the other approaches for multiclass datasets. The times of P-Delta are usually below that of G-SVM and L-SVM. In general, DPPs are 10 times below P-Delta, and 100 times below G-SVM and L-SVM. The **lower left panel** shows that DPPs times rise almost linearly and very slowly compared to P-Delta (which spends about 10 times more than DPPs) and G-SVM/L-SVM (100 times more than DPPs). In fact, from $d = 8$ (data set Pima) to $d = 60$ (Sonar), the time of DPPs *per pattern* increases only four times. With multiclass datasets (**lower right panel**), the DPPs' time per pattern and class remains almost constant with the number of inputs, while the time of P-Delta, G-SVM, and L-SVM is very high for almost all the datasets. In fact, DPPs spend less time per pattern in MNIST, with the highest number of

inputs (780), than the other approaches with the least number of inputs (Yeast, Abalone, and Nursery, 8 inputs).

The low computational cost of DPPs is expected, because the training stage in DPPs only requires: 1) the randomization of the desired outputs d_{ik} for each perceptron and training pattern (Section III-C), and 2) the calculation of the weight vector $\mathbf{w}_{i0}$ for each perceptron using (7). Both calculations were repeated for 50 randomizations of d_{ik}, but a lower number (about 10) could be used with similar results. In the test stage, DPPs just evaluate $\text{sign}(\mathbf{w}_{i0}^T\mathbf{x})$ of (1) for the n perceptrons. With more than two classes, the calculations are repeated for the N_c PPs, N_c being usually low. All the operations in DPPs are $\mathcal{O}(d)$, so that both training and test stages are very fast, and they are interesting for multidimensional and real-time problems. Finally, the number n of perceptrons does not influence very much the results: we developed experiments using n in the set $\{3, 5, 11, 15, 21, 31, 51\}$ without strong changes in the average accuracy over the datasets.

VI. Conclusion

The PP network [2] is a universal approximator which can be efficiently trained using the P-Delta rule for classification with two classes. We proposed a very simple and computationally cheap analytical approach (DPPs), an alternative to the P-Delta rule, to calculate the weights of the PPs using only the training patterns and their desired outputs. DPPs directly calculate—without iterations—the weights that globally minimize an error function measuring simultaneously the classification margin and the training error. This method does not have the problems related with slow training, parameter tuning, and getting stuck at local minima, which are usual with backpropagation and related gradient-descent approaches. DPPs are also simpler than the previous closed-form approaches found in the literature, without expensive matrix (as in [13]) or geometric calculations (e.g., the polyhedron minimal volume for a set of training patterns, as in [16], or the Tomek links as in the PLCs [17]). We also extended DPPs for multiclass problems using the OAA approach.

As opposed to other closed-form approaches [12], DPPs were evaluated in a wide collection of benchmark two-class and multiclass datasets, achieving very competitive accuracies compared to the P-Delta rule, SVM (using polynomial, Gaussian, and linear kernels), AB and BG of MLPs, LDA, MLP, and other classifiers, specially for two-class datasets. For multiclass problems, the accuracy of DPPs, and also of P-Delta, is less optimal compared to Gaussian and linear SVM, AB and BG, especially in some datasets with many patterns. In almost all the datasets, DPPs outperform the P-Delta rule. Our experimental work suggests that this reduced accuracy in multiclass problems might be caused by the limited ability of the PP network, using P-Delta or DPPs, for nonlinear classification. In fact, a single perceptron with weights calculated using the proposed expression (4), combined with a Gaussian kernel to enable nonlinear classification, achieves much better accuracy on the same datasets. DPPs are much faster than P-Delta (about 10 times) and SVM (100 times), due to their simple formulation (both for the training and test stages) and to the absence of tunable parameters (which are present in P-Delta, SVM, MLP, and almost all the classifiers). The computational complexity of DPPs is linear in the number of patterns and in the dimension of the input space. This low complexity makes DPPs very interesting for high-dimensional and real-time classification problems, and to be used as base classifiers in ensembles.

Future research includes the formulation of a kernel-based approach using the proposed expression for weight calculation, exploration of alternatives to OAA for multiclass problems, designing ensembles of perceptrons with weights calculated using (4), and proposals for an analogous method for regression problems.

Acknowledgment

The authors would like to thank the anonymous referees for their useful suggestions.

References

[1] P. Auer, H. Burgsteiner, and W. Maass, "Reducing communication for distributed learning in neural networks," in *Proc. Int. Conf. Artif. Neural Netw.*, 2002, pp. 123–128.

[2] P. Auer, H. Burgsteiner, and W. Maass, "A learning rule for very simple universal approximators consisting of a single layer of perceptrons," *Neural Netw.*, vol. 21, no. 5, pp. 786–795, Jun. 2008.

[3] W. Maass, "On the computational power of winner-take-all," *Neural Comput.*, vol. 12, no. 11, pp. 2519–2536, Nov. 2000.

[4] I. Cantador and J. R. Dorronsoro, "Parallel perceptrons, activation margins and imbalanced training set pruning," in *Pattern Recognition and Image Analysis* (Lecture Notes in Computer Science), vol. 3523. New York: Springer-Verlag, 2005, pp. 43–50.

[5] I. Cantador and J. R. Dorronsoro, "Boosting parallel perceptrons for label noise reduction in classification problems," in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach* (Lecture Notes in Computer Science), vol. 3562. New York: Springer-Verlag, 2005, pp. 586–593.

[6] I. Cantador and J. R. Dorronsoro, "Balanced boosting with parallel perceptrons," in *Computational Intelligence and Bioinspired Systems* (Lecture Notes in Computer Science), vol. 3512. New York: Springer-Verlag, 2005, pp. 208–216.

[7] A. González, I. Cantador, and J. R. Dorronsoro, "Discriminant parallel perceptrons," in *Artificial Neural Networks: Formal Models and Their Applications* (Lecture Notes in Computer Science), vol. 3697. New York: Springer-Verlag, 2005, pp. 13–18.

[8] G. Daqi, L. Hao, and C. Wei, "A mixed parallel perceptron classifier and several application problems," in *Proc. Int. Joint Conf. Neural Netw.*, Vancouver, BC, Canada, 2006, pp. 4797–4802.

[9] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[10] C. Panagiotakopoulos and P. Tsampouka, "The margitron: A generalized perceptron with margin," *IEEE Trans. Neural Netw.*, vol. 22, no. 3, pp. 395–407, Mar. 2011.

[11] T. Martinetz, K. Labusch, and D. Schneegaß, "SoftDoubleMaxMinOver: Perceptron-like training of support vector machines," *IEEE Trans. Neural Netw.*, vol. 20, no. 7, pp. 1061–1072, Jul. 2009.

[12] A. Talukder and D. Casasent, "A closed-form neural network for discriminatory feature extraction from high-dimensional data," *Neural Netw.*, vol. 14, no. 9, pp. 1201–1218, Nov. 2001.

[13] B. Liu, "Kernel-based nonlinear discriminator with closed-form solution," in *Proc. Int. Conf. Neural Netw. Signal Process.*, vol. 1. Dec. 2003, pp. 41–44.

[14] B. Liu, "Adaptive training of a kernel-based nonlinear discriminator," *Pattern Recognit.*, vol. 38, no. 12, pp. 2419–2425, 2005.

[15] C. Blake and C. Merz. (1998). *UCI Repository of Machine Learning Databases* [Online]. Available: http://archive.ics.uci.edu/ml/

[16] R. Delogu, A. Fanni, and A. Montisci, "Geometrical synthesis of MLP neural networks," *Neurocomputing*, vol. 71, nos. 4–6, pp. 919–930, Jan. 2008.

[17] H. Yenmoto, M. Kudo, and M. Shimbo, "Piecewise linear classifiers with an appropriate number of hyperplanes," *Pattern Recognit.*, vol. 31, no. 11, pp. 1627–1634, Nov. 1998.

[18] I. Tomek, "Two Modifications of CNN," *IEEE Trans. Syst., Man, Cybern.*, vol. 6, no. 11, pp. 769–772, Nov. 1976.
[19] L. Yujian, L. Bo, Y. Xinwu, F. Yaozong, and L. Houjun, "Multiconlitron: A general piecewise linear classifier," *IEEE Trans. Neural Netw.*, vol. 22, no. 2, pp. 276–289, Feb. 2011.
[20] R. Winter and B. Widrow, "MADALINE RULE II: A training algorithm for neural networks," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1. San Diego, CA, Jul. 1988, pp. 401–408.
[21] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Mateo, CA: Morgan Kaufmann, 2005.
[22] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proc. 2nd Eur. Conf. Comput. Learn. Theory*, 1995, pp. 23–37.
[23] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[24] R. Collobert, S. Bengio, and J. Mariéthoz. (2003). *Torch: A Modular Machine Learning Software Library* [Online]. Available: http://bengio.abracadoudou.com/cv/publications/ps/rr02-46.ps.gz
[25] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[26] D. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. Boca Raton, FL: CRC Press, 2006.
[27] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002.
[28] K.-A. Toh, "An error-counting network for pattern classification," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1680–1693, Mar. 2008.
[29] G.-Z. Li, T.-T. Liu, and V. S. Cheng, "Classification of brain glioma by using SVMs bagging with feature selection," in *Data Mining for Biomedical Applications* (Lecture Notes in Computer Science), vol. 3916. New York: Springer-Verlag, 2006, pp. 124–130.
[30] Y.-J. Oyang, S.-C. Hwang, Y.-Y. Ou, C.-Y. Chen, and Z.-W. Chen, "Data classification with radial basis function networks based on a novel kernel density estimation algorithm," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 225–236, Jan. 2005.
[31] P. Simard, Y. Le Cun, and J. Denker, "Efficient pattern recognition using a new transformation distance," in *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 50–58.
[32] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th Int. Conf. Document Anal. Recognit.*, vol. 2. Aug. 2003, pp. 958–963.
[33] M. Fernández-Delgado, J. Ribeiro, E. Cernadas, and S. Barro, "Fast weight calculation for kernel-based perceptron in two-class classification problems," in *Proc. Int. Joint Conf. Neural Netw.*, Barcelona, Spain, Jul. 2010, pp. 1940–1945.

Manuel Fernández-Delgado was born in A Coruña, Spain, in 1971. He received the B.S. degree in physics and the Ph.D. degree in computer science from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 1994 and 1999, respectively.

He is currently a Lecturer of computer science with the Department of Electronics and Computer Science, USC, and a part of the Intelligent Systems Group. He is a Researcher with the Centro de Investigacion en Tecnoloxías da Información, USC. His current research interests include neural computation, machine learning, and pattern recognition.

Jorge Ribeiro was born in Braga, Portugal, in 1975. He received the M.Sc. degree from the University of Minho, Guimarães, Portugal, and the Ph.D. degree in computer science from the University of Santiago de Compostela, Santiago de Compostela, Spain, in 2002 and 2011, respectively.

He is an Assistant Lecturer with the School of Technology and Management, Polytechnic Institute of Viana do Castelo, Viana do Castelo, Portugal. He is a member of the Artificial Intelligence Group, Department of Informatics, University of Minho. His current research interests include data mining, software engineering, knowledge representation, evolutionary systems, systems integration, and enterprise information systems.

Eva Cernadas was born in A Coruña, Spain, in 1969. She received the B.S. and Ph.D. degrees in physics from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 1992 and 1997, respectively.

She is a Lecturer of computer science at USC, where she is also a Researcher with the Centro de Investigacion en Tecnoloxías da Información. Her current research interests include image processing and pattern recognition, mainly applied in the food technology, robotics, and medical domains.

Senén Barro Ameneiro was born in A Coruña, Spain, in 1962. He received the B.S. and Ph.D. degrees (with honors) in physics from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 1985 and 1988, respectively.

He was an Associate Professor with the Faculty of Informatics, University of A Coruña, Corunna, Spain, until 1989. He was a Lecturer and has been a Professor of computer science with the Department of Electronics and Computer Science, USC, since 1995. From 2002 to 2010, he was a Rector of USC. He is the editor of five books and author of over 200 scientific papers in the following fields. His current research interests include signal and knowledge processing, mainly fuzzy systems, mobile robotics, and artificial neural network application and biological modeling.