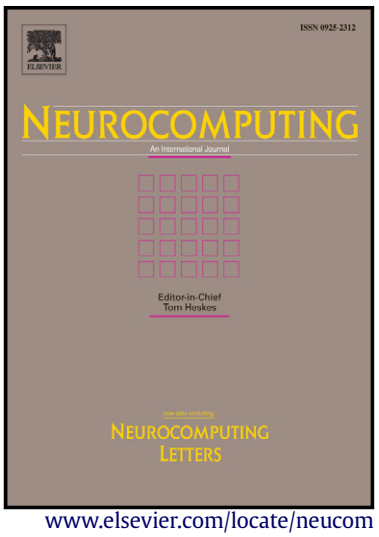

www.elsevier.com/locate/neucom

Eigenvalue decay: a new method for neural network regularization

Oswaldo Ludwig, Urbano Nunes, Rui Araujo

PII: S0925-2312(13)00833-3
DOI: http://dx.doi.org/10.1016/j.neucom.2013.08.005
Reference: NEUCOM13598

To appear in: *Neurocomputing*

Received date: 17 October 2012
Revised date: 28 April 2013
Accepted date: 14 August 2013

Cite this article as: Oswaldo Ludwig, Urbano Nunes, Rui Araujo, Eigenvalue decay: a new method for neural network regularization, *Neurocomputing*, http://dx.doi.org/10.1016/j.neucom.2013.08.005

Eigenvalue Decay: a New Method for Neural Network Regularization

Oswaldo Ludwig*, Urbano Nunes, and Rui Araujo.*

ISR-Institute of Systems and Robotics, University of Coimbra, Portugal, (email: oswaldoludwig@gmail.com)

Abstract

This paper proposes two new training algorithms for multilayer perceptrons based on evolutionary computation, regularization, and transduction. Regularization is a commonly used technique for preventing the learning algorithm from overfitting the training data. In this context, this work introduces and analyzes a novel regularization scheme for neural networks (NN) named eigenvalue decay, which aims at improving the classification margin. The introduction of eigenvalue decay led to the development of a new training method based on the same principles of SVM, and so named Support Vector NN (SVNN). Finally, by analogy with the transductive SVM (TSVM), it is proposed a transductive NN (TNN), by exploiting SVNN in order to address transductive learning. The effectiveness of the proposed algorithms is evaluated on seven benchmark datasets.

Keywords: transduction, regularization, genetic algorithm, classification margin, neural network

1. Introduction

One of the problems that occur during syntactic classifier training is called overfitting. The error on the training dataset is driven to a small value; however the error is large when new data are presented to the trained classifier. It occurs because the classifier does not learn to generalize when new situations are presented. This phenomenon is related to the classifier complexity, which can be minimized by using regularization techniques [1] and [2]. In this paper we will apply regularization to improve the classification margin, which is an effective strategy to decrease the classifier complexity, in Vapnik sense, by exploiting the geometric structure in the feature space of the training examples.

There are three usual regularization techniques for neural networks (NN): early stopping [3], curvature-driven smoothing [4], and weight decay [5]. In the early stopping criterion the labeled data are divided into training and validation datasets. After some number of iterations the NN begins to overfit the data and the error on the validation dataset begins to rise. When the validation error increases during a specified number of iterations, the algorithm stops the training section and applies the weights and biases at the minimum of the validation error to the NN. Curvature-driven smoothing includes smoothness requirements on the cost function of learning algorithms, which depend on the derivatives of the network mapping. Weight decay is implemented by including additional terms in the cost function of learning algorithms, which penalize overly high values of weights and biases, in order to control the classifier complexity, which forces the NN response to be smoother and less likely to overfit. This work introduces and analyzing a novel regularization scheme, named eigenvalue decay, aiming at improving the classification margin, as will be shown in Section 3.

In the context of some on-the-fly applications, the use of SVM with nonlinear kernels requires a prohibitive computational cost, since its decision function requires a summation of nonlinear functions that demands a large amount of time when the number of support vectors is big. Therefore, a maximal-margin neural network, [6], [7], [8], and [9], can be a suitable option for such kind of application, since it can offer a fast nonlinear classification with good generalization capacity. This work introduces a novel algorithm for maximum margin training that is based on regularization and evolutionary computing. Such method is exploited in order to introduce a transductive training method for NN.

The paper is organized as follows. In Section 2 we propose the eigenvalue decay, while the relationship between such regularization method and the classification margin is analyzed in Section 3. In Section 4 it is proposed a new maximum-margin training method based on genetic algorithms (GA) that is extended to the transductive approach in Section 5. Section 6 reports the experiments, while Section 7 summarizes some conclusions.

2. Eigenvalue decay

A multilayer perceptron (MLP) with one sigmoidal hidden layer and linear output layer is a universal approximator, because the sigmoidal hidden units of such model compose a basis of linearly independent soft functions [10]; therefore, this work focuses in such NN, whose model is given by:

$$\begin{aligned} y_h &= \varphi\left(W_1 \cdot x + b_1\right) \\ \hat{y} &= W_2^T y_h + b_2 \end{aligned} \tag{1}$$

where y_h is the output vector of the hidden layer, W_1 is a matrix whose elements are the synaptic weights that connect the input elements with the hidden neurons, W_2 is vector whose elements

*Corresponding author

are the synaptic weights that connect the hidden neurons to the output, b_1 is the bias vector of the hidden layer, b_2 is the bias of the output layer, x is the input vector, and $\varphi(\cdot)$ is the sigmoid function. The most usual objective function for MLPs is the MSE:

$$E = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

where N is the cardinality of the training dataset, y_i is the target output, $\hat{y}_i$ is the output estimated by the MLP for the input x_i belonging to the training dataset. However, in case of the usual weight decay method [11], additional terms which penalize overly high values of weights and biases are included. Therefore, the generic form for the objective function is:

$$E^* = E + \kappa_1 \sum_{w_i \in W_1} w_i^2 + \kappa_2 \sum_{w_j \in W_2} w_j^2 + \kappa_3 \sum_{b_{(1,k)} \in b_1} b_{(1,k)}^2 + \kappa_4 b_2^2 \tag{3}$$

where W_1, W_2, b_1, and b_2 are the MLP parameters, according to (1), and $\kappa_i > 0$, $i = (1 \ldots 4)$ are regularization hyperparameters. Such method was theoretically analyzed by [12], which concludes that the bounds on the expected risk of MLPs depends on the magnitude of the parameters rather than the number of parameters. Namely, in [12] the author showed that the misclassification probability can be bounded in terms of the empirical risk, the number of training examples, and a scale-sensitive version of the VC-dimension, known as the fat-shattering dimension[1], which can be upper-bounded in terms of the magnitudes of the network parameters, independently from the number of parameters[2]. In short, as regards weight-decay, [12] only shows that such method can be applied to control the capacity of the classifier space. However, the best known way to minimize the capacity of the classifier space without committing the accuracy on the training data is to maximize the classification margin, which is the SVM principle. Unfortunately, from the best of our knowledge, there is no proof that weight-decay can maximize the margin. Therefore, we propose the eigenvalue-decay, for which it is possible to establish a relationship between the eigenvalue minimization and the classification margin. The objective function of eigenvalue-decay is:

$$E^{**} = E + \kappa(\lambda_{min} + \lambda_{max}) \tag{4}$$

where λ_{min} is the smallest eigenvalue of $W_1 W_1^T$ and λ_{max} is the biggest eigenvalue of $W_1 W_1^T$.

3. Analysis on eigenvalue decay

In this section we show a relationship between eigenvalue decay and the classification margin, m_i. Our analysis requires the following lemma:

[1] See Theorem 2 of [12]

[2] See Theorem 13 of [12]

Lemma 1. [13] *Let K denotes the field of real numbers, $K^{n\times n}$ a vector space containing all matrices with n rows and n columns with entries in K, $A \in K^{n\times n}$ be a symmetric positive-semidefinite matrix, λ_{min} be the smallest eigenvalue of A, and λ_{max} be the largest eigenvalue of A. Therefore, for any $x \in K^n$, the following inequalities hold true:*

$$\lambda_{min} x^T x \leq x^T A x \leq \lambda_{max} x^T x \tag{5}$$

Proof. The upper bound on $x^T Ax$, i.e. the second inequality of (5), is well known; however, this work also requires the lower bound on $x^T Ax$, i.e. the first inequality of (5). Therefore, since this proof is quite compact, we will save a small space in this work to present the derivation of both bounds as follows:

Let $V = [v_1, \ldots, v_n]$ be the square $n \times n$ matrix whose i^{th} column is the eigenvector v_i of A, and Λ be the diagonal matrix whose i^{th} diagonal element is the eigenvalue λ_i of A; therefore, the following relations hold:

$$x^T Ax = x^T VV^{-1}AVV^{-1}x = x^T V\Lambda V^{-1}x = x^T V\Lambda V^T x \tag{6}$$

Taking into account that A is positive-semidefinite, i.e. $\forall i$, $\lambda_i \geq 0$:

$$x^T V(\lambda_{min} I)V^T x \leq x^T V\Lambda V^T x \leq x^T V(\lambda_{max} I)V^T x \tag{7}$$

where I is the identity matrix. Note that $x^T V(\lambda_{min} I)V^T x = \lambda_{min} x^T x$ and $x^T V(\lambda_{max} I)V^T x = \lambda_{max} x^T x$; therefore, substituting (6) into (7) yields (5). ■

The following theorem gives a lower and an upper bound on the classification margin:

Theorem 1. *Let m_i be the margin of the training example i, i.e. the smallest orthogonal distance between the classifier separating hypersurface and the training example i, λ_{max} be the biggest eigenvalue of $W_1 W_1^T$, and λ_{min} be the smallest eigenvalue of $W_1 W_1^T$; then, for $m_i > 0$, i.e. an example correctly classified, the following inequalities hold true:*

$$\frac{1}{\sqrt{\lambda_{max}}}\mu \leq m_i \leq \frac{1}{\sqrt{\lambda_{min}}}\mu \tag{8}$$

where

$$\mu = \min_j \left(y_i \frac{W_2^T \Gamma_j W_1 \left(x_i - x_{proj}^j\right)}{\sqrt{W_2^T \Gamma_j \Gamma_j^T W_2}} \right), \tag{9}$$

$$\Gamma_j = \begin{bmatrix} \varphi'(v_1) & 0 & \cdots & 0 \\ 0 & \varphi'(v_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \varphi'(v_n) \end{bmatrix}, \quad [v_1, v_2, \ldots, v_n]^T = W_1 \cdot x_k + b_1, \quad \varphi'(v_n) = \left.\frac{\partial \varphi}{\partial v_n}\right|_{x_{proj}^j}$$

, x_{proj}^j is the j^{th} projection of the i^{th} training example, x_i, on the separating hypersurface, as illustrated in Fig.1, and y_i is the target class of x_i.

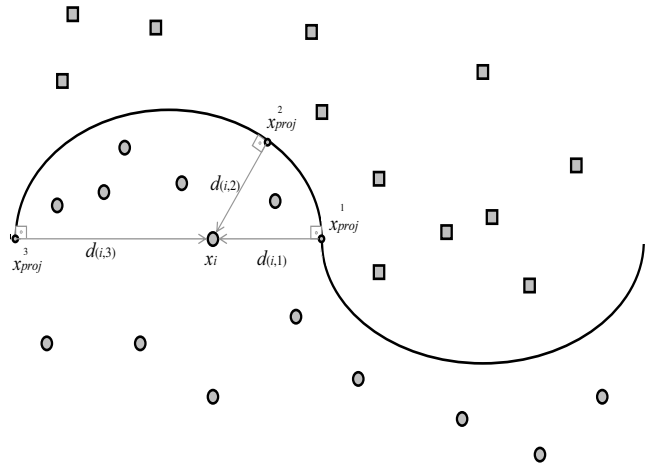

Figure 1: A feature space representing a nonlinear separating surface with the projections, x^j_{proj}, of the i^{th} training example, x_i, and examples of orthogonal distances $d_{(i,j)}$.

Proof. The first step in this proof is the calculation of the gradient of the NN output $\hat{y}$ in relation to the input vector x at the projected point, x^j_{proj}. From (1) we have:

$$\nabla\hat{y}_{(i,j)} = \left.\frac{\partial\hat{y}}{\partial x}\right|_{x^j_{proj}} = W_2^T\Gamma_j W_1 \tag{10}$$

The vector

$$\vec{p}_{(i,j)} = \frac{\nabla\hat{y}_{(i,j)}}{\left\|\nabla\hat{y}_{(i,j)}\right\|} \tag{11}$$

is normal to the separating surface, giving the direction from x_i to x^j_{proj}; therefore

$$x_i - x^j_{proj} = d_{(i,j)}\vec{p}_{(i,j)} \tag{12}$$

where $d_{(i,j)}$ is the scalar distance between x_i and x^j_{proj}. From (12) we have:

$$\nabla\hat{y}_{(i,j)}\left(x_i - x^j_{proj}\right) = d_{(i,j)}\nabla\hat{y}_{(i,j)}\vec{p}_{(i,j)} \tag{13}$$

Substituting (11) into (13) and solving for $d_{(i,j)}$, yields:

$$d_{(i,j)} = \frac{\nabla\hat{y}_{(i,j)}\left(x_i - x^j_{proj}\right)}{\left\|\nabla\hat{y}_{(i,j)}\right\|} \tag{14}$$

The sign of $d_{(i,j)}$ depends on which side of the decision surface the example, x_i, is placed. It means that an example, x_i, correctly classified whose target class is -1 corresponds to $d_{(i,j)} < 0$. On the other hand, the classification margin must be positive in case of examples correctly classified, and negative in case of misclassified examples, independently from their target classes. Therefore, the margin is defined as function of $y_i d_{(i,j)}$, where $y_i \in \{-1, 1\}$ is the target class of the i^{th} example. More specifically, the margin, m_i, is the smallest value of $y_i d_{(i,j)}$ in relation to j, that is:

$$m_i = \min_j\left(y_i d_{(i,j)}\right) \tag{15}$$

Substituting (14) in (15) yields:

$$m_i = \min_j\left(y_i\frac{\nabla\hat{y}_{(i,j)}\left(x_i - x^j_{proj}\right)}{\left\|\nabla\hat{y}_{(i,j)}\right\|}\right) \tag{16}$$

Substituting (10) in (16), yields:

$$m_i = \min_j\left(y_i\frac{W_2^T\Gamma_j W_1\left(x_i - x^j_{proj}\right)}{\sqrt{W_2^T\Gamma_j W_1 W_1^T\Gamma_j^T W_2}}\right). \tag{17}$$

Note that $W_1 W_1^T$ is a symmetric positive-semidefinite matrix, therefore, from Lemma 1, the inequalities:

$$\lambda_{min}W_2^T\Gamma_j\Gamma_j^T W_2 \le W_2^T\Gamma_j W_1 W_1^T\Gamma_j^T W_2 \le \lambda_{max}W_2^T\Gamma_j\Gamma_j^T W_2 \tag{18}$$

hold true for any Γ_j and any W_2. From (18) and (17) it is easy to derive (8). ■

Taking into account that λ_{max} and λ_{min} are the denominators of the bounds in (8), the training method based on eigenvalue decay decreases λ_{max} and λ_{min} aiming at increasing the lower and the upper bounds on the classification margin. However, eigenvalue decay does not assure, by itself, increasing the margin, because μ is function of W_1 among other NN parameters. Therefore, to evaluate the effect of eigenvalue decay on the classification margin, we performed comparative experiments with real world datasets (see Section 6). Fig. 2 illustrates the separating surface for the toy examples and Fig. 3 illustrates the margins[3] generated by NNs trained without and with eigenvalue decay. The boundary between white and colored areas represents the SVM-like classification margin, i.e. for input data belonging to the yellow area, the NN model outputs $0 \le \hat{y}_i < 1$, while for input data belonging to the green area, the model outputs $0 > \hat{y}_i > -1$. The boundary between colored areas represents the separating surface. The training methods proposed in this paper are similar to SVM, i.e. the data which lie into the colored areas, or fall on the wrong side of the separating surface, increase a penalty term. The algorithm minimizes the penalty term in such a way to move the colored area, and so the separating surface, away from the training data. Therefore, the larger is the colored area, i.e. the smaller the eigenvalues, the larger the distance between the training data and the separating surface.

4. Maximum-margin Training by GA

Theorem 1 allows us to propose a maximal-margin training method quite similar to SVM [14], in the sense that the proposed method also minimizes values related with the parameters of the classifier model, in order to maximize the margin,

[3]The margins were generated by a dense grid of points. The output of the trained NN is calculated for each point. If the output is $-1 < y < 1$, the point receives a colored pixel.

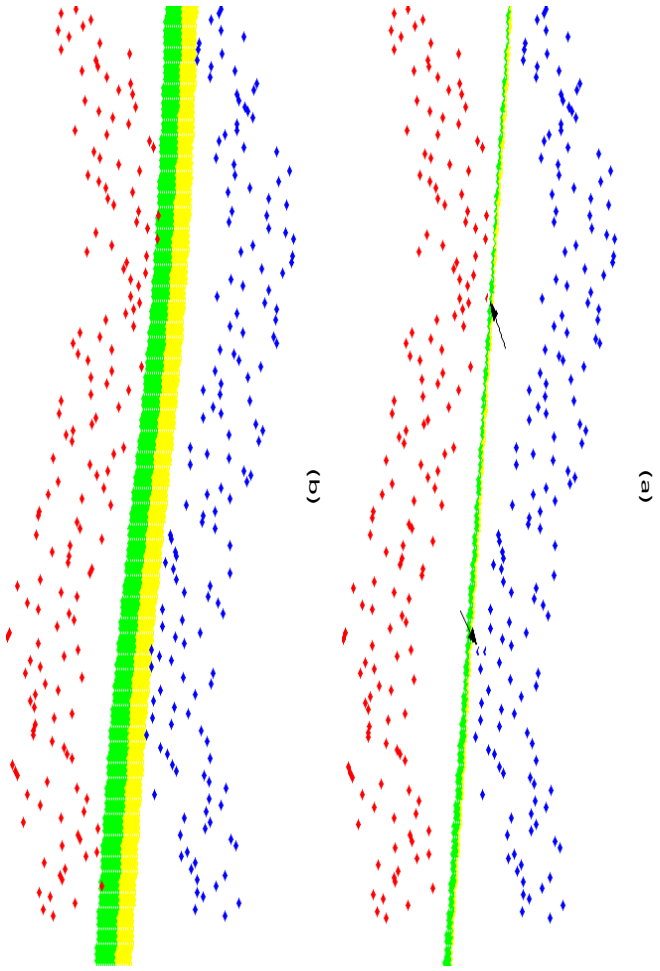

Figure 2: First toy example on the effect of eigenvalue decay on the margin: (a) NN trained without eigenvalue decay, (b) NN trained with eigenvalue decay. The boundary between white and colored areas represents the SVM-like classification margin, i.e. for input data belonging to the yellow area, the NN model outputs $0 \leq \hat{y}_i < 1$, while for input data belonging to the green area, the model outputs $0 > \hat{y}_i > -1$.

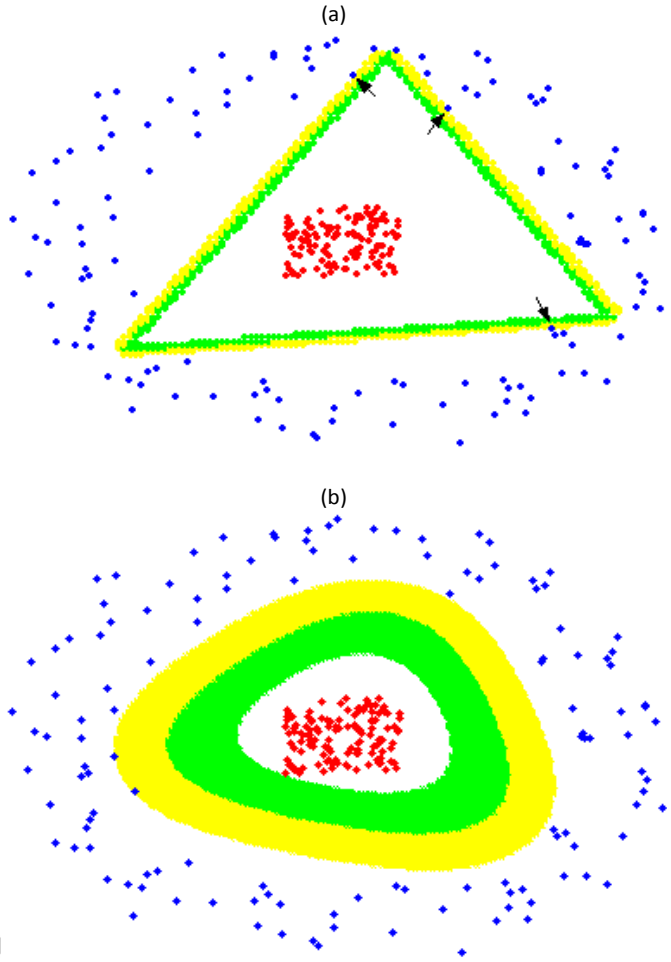

Figure 3: Second toy example: (a) NN trained without eigenvalue decay, (b) NN trained with eigenvalue decay.

allowing the minimization of the classifier complexity without committing the accuracy on the training data.

The main idea of our method is not only to avoid nonlinear SVM kernels, in such a way as to offer a faster nonlinear classifier, but also to be based on the maximal-margin principle; moreover, the proposed method is more suitable for on-the-fly applications, such as object detection [15], [16]. The SVM decision function is given by:

$$c(x) = \mathrm{sgn}\left(\sum_{i=1}^{N_{sv}} y_i \alpha_i K(x_i, x) + b\right) \tag{19}$$

where α_i and b are SVM parameters, (x_i, y_i) is the i^{th} support vector data pair, $\mathrm{sgn}(\cdot)$ is 1 if the argument is greater than zero and -1 if it is less than zero, and $K(\cdot,\cdot)$ is a non-linear kernel function, i.e. the algorithm fits the maximum-margin hyperplane in a transformed feature space, in order to enable a non-linear classification. Notice that (19) requires a large amount of time when the number of support vectors, N_{sv}, is big. This fact motivated this new SVM-like training method for NN, named Support Vector NN (SVNN), here proposed.

In order to better understand our method, it is convenient to take into account the soft margin SVM optimization problem, as follows:

$$\min_{w,\xi_i}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i\right) \tag{20}$$

subject to

$$\forall i\,|y_i\,(w \cdot x_i - b) \geq 1 - \xi_i \tag{21}$$

$$\forall i\,|\xi_i \geq 0 \tag{22}$$

where w and b compose the separating hyperplane, C is a constant, y_i is the target class of the i^{th} training example, and ξ_i are slack variables, which measure the degree of misclassification of the vector x_i. The optimization is a trade off between a large margin ($\min \|w\|^2$), and a small error penalty ($\min C\sum_{i=1}^{N}\xi_i$).

We propose to train the NN by solving the similar optimization problem:

$$\min_{W_1,\xi_i}\left(\lambda_{min} + \lambda_{max} + \frac{C_1}{N}\sum_{i=1}^{N}\xi_i\right) \tag{23}$$

subject to

$$\forall i \, | y_i \hat{y}_i \geq 1 - \xi_i \tag{24}$$

$$\forall i \, | \xi_i \geq 0 \tag{25}$$

where $\hat{y}$ is given by (1), C_1 is a regularization hyperparameter, y_i is the target class of the i^{th} training example, and ξ_i are also slack variables, which measure the degree of misclassification of the vector x_i.

The constrained optimization problem (23)-(25) is replaced by the equivalent unconstrained optimization problem (26) [17], that has the discontinuous objective function Φ, which disables the gradient-based optimization methods; therefore, a real-coded GA is applied to solve (26), using Φ as fitness function [18].

$$\min_{W_1, W_2, b_1, b_2} \Phi \tag{26}$$

where

$$\Phi = \lambda_{min} + \lambda_{max} + \frac{C_1}{N} \sum_{i=1}^{N} H(y_i \hat{y}_i) \tag{27}$$

and $H(t) = \max(0, 1 - t)$ is the Hinge loss.

Note that the last term of (27) penalizes models whose estimated outputs do not fit the constraint $y_i \hat{y}_i \geq 1$, in such a way as to save a *minimal margin*, while the minimization of the first two terms of (27) aims at the enlargement of such *minimal margin* by eigenvalue decay, as suggested by Theorem 1.

Algorithm 1 details the proposed optimization process. The chromosome of each individual is coded into a vertical vector composed by the concatenation of all the columns of W_1 with W_2, b_1, and b_2. The algorithm starts by randomly generating the initial population of N_{pop} individuals in a uniform distribution, according to the Nguyen-Widrow criterion [19]. During the loop over generations the fitness value of each individual is evaluated on the training dataset, according to (26) and (27). Then, the individuals are ranked according to their fitness values, and the crossover operator is applied to generate new individuals by randomly selecting the parents by their ranks, according to the random variable $p \in [1, N_{pop}]$ proposed in our previous work [20]:

$$p = \left(N_{pop} - 1\right) \frac{e^{a\vartheta} - 1}{e^a - 1} + 1 \tag{28}$$

where $\vartheta \in [0, 1]$ is a random variable with uniform distribution and $a > 0$ sets the selective pressure, more specifically, the larger a, the larger the probability of low values of p, which are related to high-ranked individuals.

5. Transductive Neural Networks

This section deals with transduction, a concept in which no general decision rule is inferred. Differently from inductive inference, in the case of transduction the inferred decision rule aims only at the labels of the unlabeled testing data.

Algorithm 1 Maximal Margin Training by GA

Input: X, y: matrices with N training datapairs;
n_{neu}: number of hidden neurons;
C_1 : regularization hyperparameter;
a: selective pressure;
max_{gener}: maximum number of generations;
N_{pop}: population size
Output: W_1, W_2, b_1, and b_2: NN parameters
generate a set of N_{pop} chromosomes, $\{Cr\}$, for the initial population, taking into account the number of input elements and n_{neu}; therefore, each chromosome is a vertical vector $Cr = [w_1, \ldots, w_{n_w}, b_1, \ldots, b_{n_b}]^T$ containing all the N_g synaptic weights and biases randomly generated according to the Nguyen-Widrow criterion [19];
for *generation* $= 1 : max_{gener}$ **do**
  evaluating the population:
  for *ind* $= 1 : N_{pop}$ **do**
    rearrange the genes, Cr^{ind}, of individual *ind*, in order to compose the NN parameters W_1, W_2, b_1, and b_2.
    for $i = 1 : N$ **do**
      calculate $\hat{y}_i$, according to (1), using the weights and biases of individual *ind*;
    end for
    calculate Φ for the individual *ind*, according to (26), using y and the set of NN outputs $\{\hat{y}_i\}$ previously calculated;
    $\Phi_{ind} \leftarrow \Phi$: storing the fitness of individual *ind*;
  end for
  rank the individuals according to their fitness Φ_{ind};
  store the genes of the best individual in Cr^{best};
  performing the crossover:
  $k \leftarrow 0$;
  for *ind* $= 1 : N_{pop}$ **do**
    $k \leftarrow k + 1$;
    randomly selecting the indexes of parents by using the asymmetric distribution proposed in [20], and also applied in [21]:
    $\vartheta_j \leftarrow$ random number $\in [0, 1]$ with uniform distribution, $j = (1, 2)$;
    $parent_j \leftarrow round\left(\left(N_{pop} - 1\right) \frac{e^{a\vartheta_j} - 1}{e^a - 1} + 1\right)$, $j = (1, 2)$;
    assembling the chromosome Cr_k^{son} :
    for $n = 1 : N_g$ **do**
      $\eta \leftarrow$ random number $\in [0, 1]$ with uniform distribution;
      $Cr_{(k,n)}^{son} \leftarrow \eta Cr_{(parent_1,n)} + (1 - \eta) Cr_{(parent_2,n)}$: calculating the n^{th} gene to compose the chromosome of the k^{th} individual of the new generation, by means of weighted average;
    end for
  end for
end for
rearrange the genes of the best individual, Cr^{best}, in order to compose the NN parameters W_1, W_2, b_1, and b_2.

The SVM-like training method, introduced in the previous section, can be exploited to address transductive learning. Therefore, we propose the transductive NN (TNN), which is similar to the transductive SVM (TSVM) [14]. The transductive algorithm takes advantage of the unlabeled data similarly to the inductive semi-supervised learning algorithm. However, differently from the inductive semi-supervised learning, transduction is based on the Vapnik principle, which states that when trying to solve some problem, one should not solve a more difficult problem, such as the induction of a general decision rule, as an intermediate step.

The proposed TNN accomplishes transduction by finding those test labels for which, after training a NN on the combined training and test datasets, the margin on the both datasets is maximal. Therefore, similarly to TSVM, TNN exploits the geometric structure in the feature vectors of the test examples, by taking into account the principle of low density separation. Such principle assumes that the decision boundary should lie in a low-density region of the feature space, because a decision boundary that cuts a data cluster into two different classes is not in accordance with the cluster assumption, which can be stated as follows: if points are in the same data cluster, they are likely to be of the same class.

The TNN training method can be easily implemented by including in (26) an additional term that penalizes all the unlabeled data which are near to the decision boundary, in order to place the decision boundary away from high-density regions. Therefore, the new optimization problem is:

$$\min_{W_1,W_2,b_1,b_2} \Phi^* \tag{29}$$

where

$$\Phi^* = \lambda_{min} + \lambda_{max} + \frac{C_1}{N}\sum_{i=1}^{N} H(y_i\hat{y}_i) + \frac{C_2}{N_u}\sum_{j=1}^{N_u} H(|\hat{y}_j|), \tag{30}$$

C_1 and C_2 are constants, $\hat{y}_j$ is the NN output for the unlabeled data x_j, and N_u is the cardinality of the unlabeled dataset. Notice that, the operator $|\cdot|$ makes this additional term independent of the class assigned by the NN for the unlabeled example, i.e. independent from the signal of $\hat{y}_j$, since we are interested only in the distance from the unlabeled data to the decision boundary.

In order to illustrate the effect of the last term of (30), we introduce two toy examples which enable a comparative study on the decision boundaries generated by SVNN and TNN, as can be seen in Figs. 4 and 5, where circles represent training data and points represent testing (unlabeled) data.

Note that both toy examples are in accordance with the cluster assumption, i.e. there are low-density regions surrounding data clusters whose elements belong to the same class. TNN places the separating-surface along such low-density regions (see Fig. 4(b)), in order to increase the absolute value of the margin of the unlabeled data, in such a way as to decrease the last term of (30).

Empirically, it is sometimes observed that the solution to (30) is unbalanced, since it is possible to decrease the last term of (30) by placing the separating-surface away from all the testing

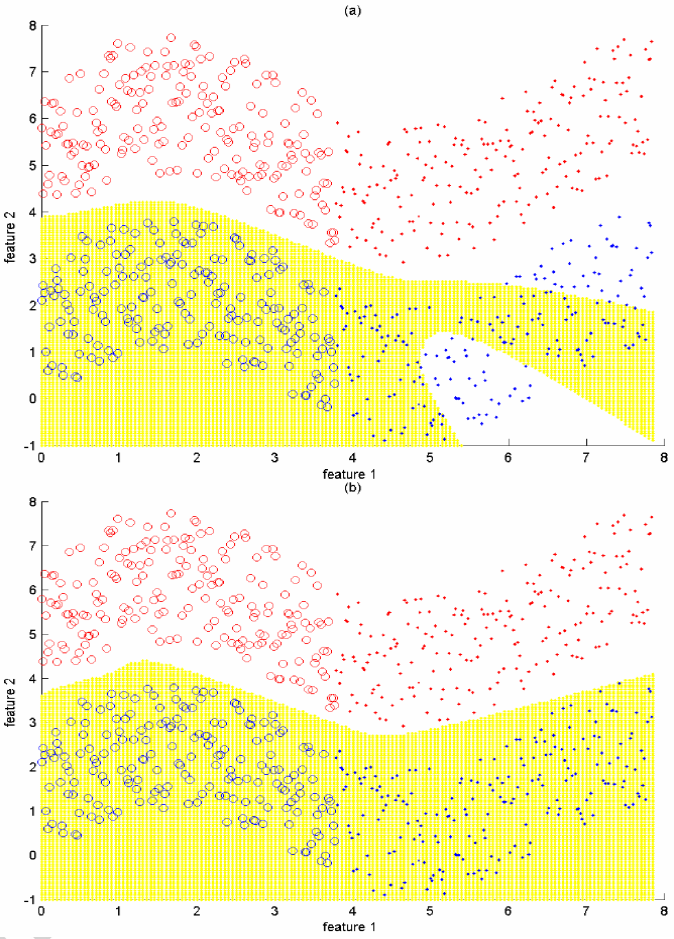

Figure 4: Separating surfaces generated by two NNs with 5 hidden neurons. Circles represent training data and points represent testing (unlabeled) data: (a) NN trained by SVNN, (b) NN trained by TNN.

instances, as can be seen in Fig. 6. In this case, all the testing instances are predicted in only one of the classes. Such problem can also be observed in case of TSVM, for which a heuristic solution is to constrain the predicted class proportion on the testing data, so that it is the same as the class proportion on the training data. This work adopts a similar solution for TNN, by including in (30) a term that penalizes models whose predicted class proportion on the testing data is different from the class proportion on the training data. Therefore, we rewrite (30) as:

$$\begin{aligned}\Phi^* = \lambda_{min} + \lambda_{max} + \tfrac{C_1}{N}\textstyle\sum_{i=1}^{N} H(y_i\hat{y}_i) + \tfrac{C_2}{N_u}\sum_{j=1}^{N_u} H(|\hat{y}_j|) \\ +C_3\left|\tfrac{1}{N}\textstyle\sum_{i=1}^{N} y_i - \tfrac{1}{Nu}\sum_{j=1}^{N_u} \mathrm{sgn}\left(\hat{y}_j\right)\right|\end{aligned} \tag{31}$$

where C_3 is a penalization coefficient. Fig. 7 shows the separating-surface of TNN after the inclusion of the last term of (31).

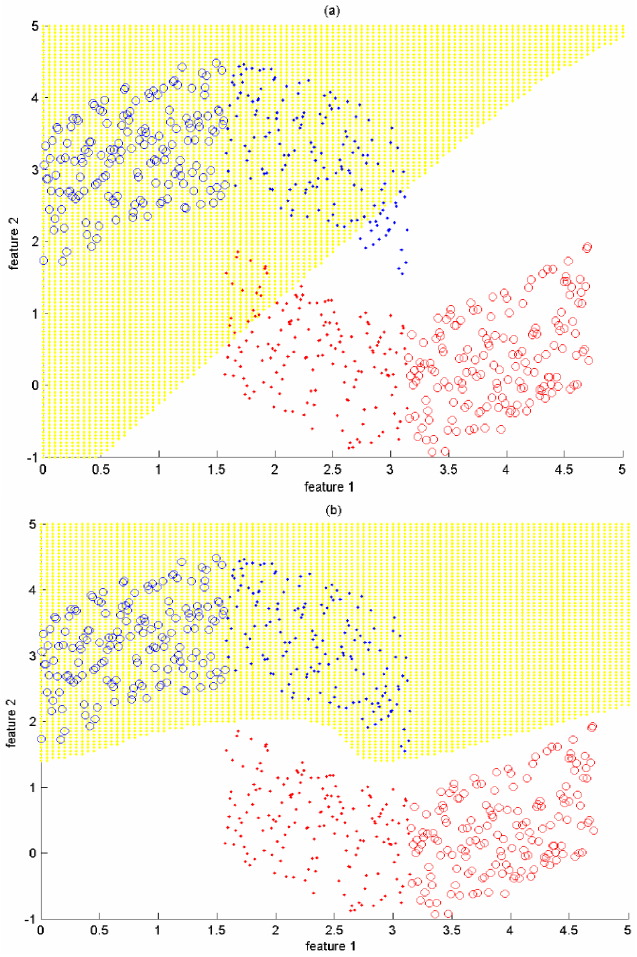

Figure 5: Separating surfaces generated by two NNs with 4 hidden neurons. Circles represent training data and points represent testing (unlabeled) data: (a) NN trained by SVNN, (b) NN trained by TNN.

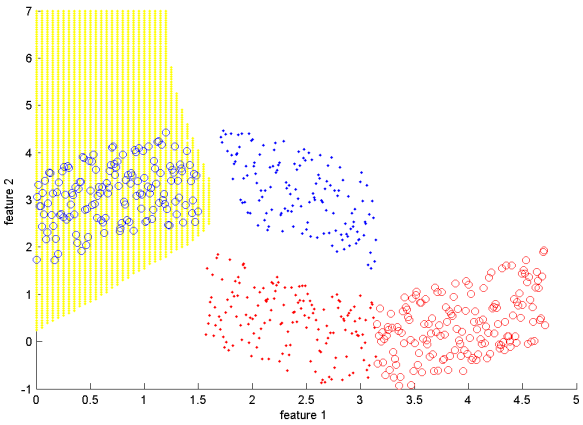

Figure 6: Toy experiment using TNN trained without the last term of (31). Circles represent training data and points represent testing (unlabeled) data.

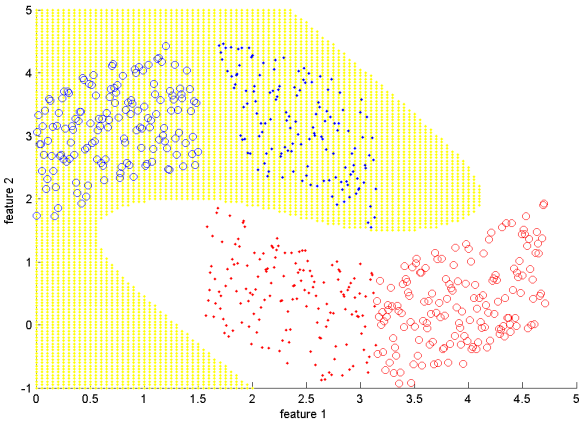

Figure 7: Toy experiment using TNN trained with the last term of (31). Circles represent training data and points represent testing (unlabeled) data.

6. Experiments

In this section our methods are evaluated by means of experiments in three UCI benchmark datasets[4] and four datasets from [22][5]. Table 1 details the applied datasets.

Table 1: Datasets used in the experiments

Dataset	Attributes	# data train	# data test	Total
Breast-cancer	9	200	77	277
Haberman	3	153	153	306
Hepatitis	19	77	78	155
BCI	117	200	200	400
Digit1	241	750	750	1500
g241c	241	750	750	1500
Text	11960	750	750	1500

The highly unbalanced datasets: Breast-cancer, Haberman, and Hepatitis, were introduced in our experimental analysis in order to verify the behavior of the optimization algorithms of transductive methods when working under the constraint on the predicted class proportion, i.e. the last term of (31). The other datasets are usually applied to evaluate semi-supervised learning algorithms. Each dataset was randomly divided into 10-folds, thus, all the experimental results were averaged over 10 runs.

The datasets were randomly divided into 10-folds, in order to average the results over 10 runs. Each fold contains all the data divided into two subsets: half for training and half for testing. For each training dataset, i.e. half of the data, it was performed 10-folds cross-validation to set the classifier parameters. Therefore, the parameter setting does not take into account information from the testing dataset. In the case of SVM, the soft-margin parameter C and the RBF parameter γ (in the case of RBF kernel) were chosen in the set $\left\{10^{-3}, 10^{-2}, \ldots, 10^{3}\right\}$. In case of SVNN and TNN, the parameters C_1, C_2, and C_3 were

[4] http://archive.ics.uci.edu/ml/datasets/
[5] http://olivier.chapelle.cc/ssl-book/benchmarks.html

Table 2: Number of hidden neurons of both SVNN and TNN

Breast-cancer	Haberman	Hepatitis	BCI	Digit1	g241c	Text
2	4	2	2	2	2	1

chosen in the set $\left\{10^2, 10^3, 10^4, 10^5\right\}$. Table 2 gives the averaged number of hidden neurons adopted after the cross-validation.

In our experiments SVNN and TNN were compared with NNs trained by the usual Levenberg-Maquardt (LM), as well as, SVM and TSVM[6]. Since SVNN and TNN can perform nonlinear classification, we also evaluated the performance of SVM and TSVM with RBF kernel. In order to verify the capability of transductive algorithms in learning from few labeled data, all the algorithms were trained by using only 10 labeled training points, as well as, all the training data. In both cases, the transductive algorithms made use of all the testing (unlabeled) data. Regarding the GA, the selective pressure was set to $a = 6$ and the population $N_{pop} = 3000$.

Tables 3-6 summarize the experimental results. Equations (32), (33), and (34) define the three indexes adopted to assess the learning performance, i.e. accuracy (*Acc*), balanced error rate (*BER*), and a measure of precision named positive predictive value (*PPV*).

$$Acc = \frac{TP + TN}{(np + nn)} \tag{32}$$

$$BER = \frac{1}{2}\left(\frac{FP}{nn} + \frac{FN}{np}\right) \tag{33}$$

$$PPV = \frac{TP}{(TP + FP)} \tag{34}$$

where TP is the number of positive examples correctly classified, TN is the number of negative examples correctly classified, FP is the number of negative examples classified as positive, FN is the number of positive examples classified as negative, np is the number of positive examples, and nn is the number of negative examples.

Tables 7 and 8 report the training and testing time in seconds, averaged on all the cross-validation runs.

In order to evaluate the influence of eigenvalue decay on the performance of SVNN, two sets of experiments were performed. In the first set of experiments a SVNN was trained by minimizing (27) without the first term; therefore, this model was named SVNN$-\lambda_{min}$. In the second set of experiments a SVNN was trained by minimizing (27) without the first two terms; hence, this model was named SVNN$-\lambda_{min} - \lambda_{max}$. Both models were evaluated on all the datasets of Table 1. Moreover, it was investigated the influence of the term about class proportion on the accuracy of TNN. To do so, a TNN was trained by minimizing only the first four terms of (31); hence, this model was named TNN$-C_3$. The results are summarized in Table 9.

As regards the inductive training methods, SVNN had the best performance and in the majority of the evaluated data sets by using only 10 labeled data. In the case of the Haberman dataset, all the algorithms fail, predicting all the testing data in

[6]http://svmlight.joachims.org/

Table 3: Accuracy (*Acc*), balanced error rate (*BER*), and positive predictive value (*PPV*) of inductive methods with 10 labeled training points.

	SVNN	NN-LM	SVM-l	SVM-rbf
		Breast Cancer dataset		
Acc	**62.86±9.16**	60.26±9.42	61.69±7.83	62.73±5.25
BER	**38.92±6.65**	42.24±8.11	55.04±3.81	42.15±4.55
PPV	**36.37±10.51**	33.24±9.82	14.96±8.94	34.34±5.71
		Haberman dataset		
Acc	26.60±0.00	26.60±0.00	26.60±0.00	26.60±0.00
BER	50.00±0.00	50.00±0.00	50.00±0.00	50.00±0.00
PPV	26.47±0.00	26.47±0.00	26.47±0.00	26.47±0.00
		Hepatitis dataset		
Acc	61.03±3.44	58.33±3.82	**61.79±3.97**	53.59±4.50
BER	**42.99±2.87**	44.21±2.84	47.25±2.92	54.94±4.04
PPV	**26.36±2.29**	24.97±2.42	23.22±3.16	16.32±3.36
		BCI dataset		
Acc	**51.70±4.18**	51.00±5.26	50.15±5.11	50.90±5.67
BER	**48.30±4.18**	49.00±5.26	49.85±5.11	49.10±5.67
PPV	**51.20±3.49**	50.50±4.35	49.65±5.08	50.40±4.26
		Digit1 dataset		
Acc	**74.97±4.63**	73.76±5.70	69.47±4.22	73.81±4.53
BER	**25.54±4.05**	26.41±4.98	30.93±4.06	26.32±4.45
PPV	**96.73±3.68**	77.32±4.11	79.83±3.52	76.20±3.65
		g241c dataset		
Acc	**61.68±3.74**	55.84±3.92	57.70±3.17	55.76±3.82
BER	**38.32±3.74**	44.16±3.92	42.30±3.17	44.24±3.82
PPV	**61.61±2.91**	55.77±3.52	57.63±2.52	55.69±3.35
		Text dataset		
Acc	**57.68±3.89**	57.15±4.07	55.17±3.91	54.33±4.02
BER	**42.83±3.80**	43.71±4.01	44.21±3.88	44.30±3.97
PPV	**57.40±3.53**	56.86±3.72	56.52±3.06	55.19±3.32

Table 4: Accuracy (*Acc*), balanced error rate (*BER*), and positive predictive value (*PPV*) of transductive methods with 10 labeled training points.

	TNN	TSVM-l	TSVM-rbf
	Breast Cancer dataset		
Acc	71.43±1.83	74.03±0.00	74.03±0.00
BER	**30.74±1.53**	50.00±0.00	50.00±0.00
PPV	**46.41±2.88**	25.97±0.00	25.97±0.00
	Haberman dataset		
Acc	**75.95±2.49**	73.40±0.00	73.40±0.00
BER	**33.47±2.31**	50.00±0.00	50.00±0.00
PPV	**55.45±3.27**	26.47±0.00	26.47±0.00
	Hepatitis dataset		
Acc	**78.72±2.21**	78.08±0.00	78.08±0.00
BER	**34.03±2.10**	50.00±0.00	50.00±0.00
PPV	**47.97±3.38**	20.51±0.00	20.51±0.00
	BCI dataset		
Acc	**52.20±3.42**	50.90±3.55	51.00±4.85
BER	**47.80±3.42**	49.10±3.55	49.00±4.85
PPV	**51.95±1.58**	50.65±1.59	50.75±2.23
	Digit1 dataset		
Acc	**82.43±4.02**	80.18±4.26	82.20±3.97
BER	**17.84±3.94**	21.58±4.12	18.33±4.06
PPV	**92.77±6.29**	90.12±5.01	91.86±4.34
	g241c dataset		
Acc	**77.79±3.18**	76.11±3.29	75.29±3.02
BER	**22.21±3.18**	23.89±3.29	24.71±3.02
PPV	**77.74±2.58**	76.06±2.55	75.24±2.28
	Text dataset		
Acc	68.82±3.67	**69.03±3.22**	65.44±4.01
BER	32.10±3.21	**31.72±3.38**	35.06±3.43
PPV	68.53±3.21	**68.95±3.05**	64.63±4.08

Table 5: Accuracy (*Acc*), balanced error rate (*BER*), and positive predictive value (*PPV*) of inductive methods with all the labeled training points.

	SVNN	NN-LM	SVM-l	SVM-rbf
		Breast Cancer dataset		
Acc	**73.77±4.11**	65.58±4.61	72.72±3.83	73.64±4.61
BER	**28.26±3.52**	42.22±4.27	37.04±3.71	37.22±5.46
PPV	**49.63±5.23**	35.93±4.88	47.21±3.79	49.09±5.97
		Haberman dataset		
Acc	**75.29±2.88**	74.12±3.19	72.03±2.76	74.51±3.16
BER	**32.05±2.67**	33.51±3.05	35.44±2.32	32.87±2.84
PPV	**53.39±3.01**	51.13±4.22	47.25±3.17	51.86±3.87
		Hepatitis dataset		
Acc	**71.79±2.85**	68.97±3.11	64.23±2.34	71.15±3.34
BER	**30.26±2.81**	32.93±2.95	32.67±2.12	30.36±3.20
PPV	**38.96±3.20**	35.67±4.14	33.06±2.59	38.37±4.08
		BCI dataset		
Acc	**80.15±2.23**	72.55±2.83	77.10±2.12	79.30±2.60
BER	**19.85±2.23**	27.45±2.83	22.90±2.12	20.70±2.60
PPV	**79.99±1.95**	72.35±2.00	76.92±1.69	79.13±2.21
		Digit1 dataset		
Acc	**95.33±2.61**	93.64±2.72	90.77±2.45	94.98±2.13
BER	6.42±2.32	7.72±2.68	10.82±2.29	**5.12±2.09**
PPV	**88.21±2.51**	87.75±2.31	75.25±3.47	87.76±2.51
		g241c dataset		
Acc	**79.60±2.64**	68.88±2.83	78.79±2.67	78.66±2.81
BER	**20.40±2.64**	31.12±2.83	21.21±2.67	21.34±2.81
PPV	**79.55±2.26**	68.82±1.82	78.74±2.23	78.32±2.07
		Text dataset		
Acc	85.57±1.87	75.13±2.22	**86.84±1.82**	78.73±2.12
BER	15.49±1.76	26.55±2.13	**15.32±1.68**	23.05±1.64
PPV	84.92±1.48	74.16±2.06	**86.57±1.55**	77.35±1.82

Table 6: Accuracy (*Acc*), balanced error rate (*BER*), and positive predictive value (*PPV*) of transductive methods with all the labeled training points.

	TNN	TSVM-l	TSVM-rbf
	Breast Cancer dataset		
Acc	**75.58±1.41**	72.73±1.17	74.81±1.44
BER	**26.99±1.12**	35.33±1.09	33.21±1.32
PPV	**52.31±1.68**	47.52±1.58	51.55±2.17
	Haberman dataset		
Acc	**76.01±2.13**	73.40±0.00	73.40±0.00
BER	**30.81±2.10**	50.00±0.00	50.00±0.00
PPV	**54.68±2.91**	26.47±0.00	26.47±0.00
	Hepatitis dataset		
Acc	73.85±2.06	**80.13±1.22**	79.49±1.36
BER	28.72±1.89	**24.45±1.18**	25.13±1.24
PPV	41.48±2.62	**51.18±1.65**	50.01±2.33
	BCI dataset		
Acc	**80.60±2.10**	80.10±2.07	80.30±2.14
BER	**19.40±2.10**	19.90±2.07	19.70±2.14
PPV	**80.44±1.86**	79.94±1.80	80.14±1.88
	Digit1 dataset		
Acc	**95.44±2.12**	94.43±2.04	92.37±2.21
BER	**5.47±2.25**	6.06±1.94	8.24±2.18
PPV	**88.78±2.54**	88.12±2.96	87.55±2.64
	g241c dataset		
Acc	**82.86±2.24**	81.93±2.52	81.88±2.21
BER	**17.14±2.24**	18.07±2.52	18.12±2.21
PPV	**82.82±2.12**	81.89±2.31	81.84±2.02
	Text dataset		
Acc	86.53±1.21	**87.12±1.12**	79.61±1.48
BER	14.96±1.18	**14.24±1.03**	21.33±1.39
PPV	85.25±1.32	**87.16±2.14**	78.22±1.43

the same class (see the value of $BER = 50\%$ in Table 3). By using all the training data, SVNN only was less accurate than

Table 7: Training and testing time, in seconds, of NNs with all the labeled training points.

	SVNN	NN-LM	TNN
	Breast Cancer dataset		
Train	24.53	12.55	74.28
Test	0.01	0.01	0.01
	Haberman dataset		
Train	69.33	14.12	78.37
Test	0.01	0.01	0.01
	Hepatitis dataset		
Train	29.56	16.76	65.54
Test	0.01	0.01	0.01
	BCI dataset		
Train	185.34	32.58	214.76
Test	0.01	0.01	0.01
	Digit1 dataset		
Train	684.95	92.67	996.47
Test	0.02	0.02	0.02
	g241c dataset		
Train	673.65	89.73	989.75
Test	0.02	0.02	0.02
	Text dataset		
Train	990.35	193.18	1287.83
Test	0.06	0.06	0.06

Table 8: Training and testing time, in seconds, of SVMs with all the labeled training points.

	SVM-l	SVM-rbf	TSVM-l	TSVM-rbf
		Breast Cancer dataset		
Train	0.02	0.13	0.28	1.32
Test	0.01	0.02	0.01	0.03
		Haberman dataset		
Train	37.39	52.45	368.71	87.42
Test	0.01	0.05	0.01	0.05
		Hepatitis dataset		
Train	1.23	241.36	22.32	287.22
Test	0.01	0.12	0.01	0.12
		BCI dataset		
Train	5.45	10.36	9.22	75.36
Test	0.01	0.08	0.01	0.06
		Digit1 dataset		
Train	2.53	19.74	838.29	1442.94
Test	0.04	2.56	0.06	4.16
		g241c dataset		
Train	1.56	32.04	84.19	242.90
Test	0.03	1.59	0.06	3.92
		Text dataset		
Train	8.36	3764.61	27.98	5138.61
Test	0.43	3.05	0.63	3.16

SVM in the Text dataset (see Table 5). We believe that this fact is due to the high-dimensional feature space of Text dataset, since such fact can favor linear classifiers, such as the linear SVM.

As regards the transductive training methods, TSVM and TSVM-rbf predicted all the testing data of the UCI datasets in the majority class when using only 10 labeled data, i.e. the constraint on the predicted class proportion was violated (see the value of $BER = 50\%$ in the first three rows of Table 4). Therefore, TNN was the best approach for all the datasets, excepting the Text dataset, for which the linear TSVM was the best approach. By using all the training data, TNN had the best values of accuracy, *BER*, and *PPV* in five of the seven datasets.

As regards the training and testing time, SVM was, in most of the experiments, less expensive in training than the proposed methods; however, the testing time reveals the main advantage of SVNN and TNN, which can perform nonlinear classification

Table 9: Performance of SVNN without eigenvalue decay and TNN without the term about class proportion.

	SVNN$-\lambda_{min}$	SVNN-λ_{min} $-$ λ_{max}	TNN$-C_3$
	Breast Cancer dataset		
Acc	71.04±4.26	68.96±4.12	74.03±0.00
BER	30.23±4.38	33.51±4.19	50.00±0.00
PPV	46.05±5.33	43.14±5.06	25.97±0.00
	Haberman dataset		
Acc	75.16±2.65	73.86±2.61	73.40±0.00
BER	33.25±2.44	34.12±2.73	50.00±0.00
PPV	53.36±2.88	50.64±3.07	26.47±0.00
	Hepatitis dataset		
Acc	70.51±2.77	67.95±2.32	73.08±1.36
BER	32.85±2.64	34.12±2.28	29.07±1.34
PPV	36.87±2.82	34.46±3.04	40.58±2.11
	BCI dataset		
Acc	80.05±2.16	73.50±2.72	80.10±2.31
BER	19.95±2.16	26.50±2.72	19.90±2.31
PPV	79.89±1.88	73.30±1.97	79.94±2.01
	Digit1 dataset		
Acc	94.67±2.71	90.93±2.63	95.33±2.20
BER	5.83±2.75	10.16±2.50	5.72±2.28
PPV	87±2.65	84.04±3.06	89.24±3.21
	g241c dataset		
Acc	78.93±2.74	73.20±2.83	82.80±2.08
BER	21.07±2.74	26.80±2.83	17.20±2.08
PPV	78.88±2.30	73.14±2.03	82.76±1.96
	Text dataset		
Acc	85.33±1.84	82.40±2.01	86.67±1.41
BER	15.12±1.67	19.02±2.12	16.32±1.29
PPV	84.54±1.58	83.13±2.00	86.84±1.16

a few hundred times faster than SVM with nonlinear kernels, as can be seen, for instance, in the fifth row of Table 7. In this case, TSVM has 231 support-vectors, while the TNN has only two hidden neurons; therefore, taking into account that Digit1 dataset has 241 attributes, from the models (1) and (19) it is possible to realize that the decision function of TSVM requires the calculation of 56133 products, 55672 sums, and 232 nonlinear functions, while the decision function of TNN only requires the calculation of 484 products, 487 sums, and 2 nonlinear functions. Such fact is especially relevant in applications such as on-the-fly object detection, in which each image frame has to be scanned by a sliding window, generating several thousands of cropped images to be classified.

By comparing Tables 9 and 5, it is possible to verify the positive influence of eigenvalue decay on the performance of SVNN. Table 9 also reveals the importance of the term about class proportion on the performance of TNN. Note that, TNN$-C_3$ is unsatisfactory in classifying the first two datasets, i.e. TNN predicted all the testing data of datasets Breast Cancer and Haberman in the majority class (see the value of $BER = 50\%$ in the last cells of the first two rows of Table 9).

7. Conclusion

The analysis presented in this paper indicates that by applying eigenvalue decay it is possible to increase the classification margin, which improves the generalization capability of NNs. The introduction of eigenvalue decay allowed the synthesis of two novel SVM-like training methods for NNs, including a transductive algorithm. These methods are suitable options for a faster non-linear classification, by avoiding the time expensive decision-function of non-linear SVMs, which may hinder on-the-fly applications, such as pedestrian detection (e.g. see Section 4.2 of [15]). The experiments indicate that, regarding the classification accuracy, SVNN and TNN are similar to non-linear SVM and TSVM; however, regarding the testing time, the proposed methods were significantly faster than non-linear SVMs. The experiments also indicate that TNN can take advantage of unlabeled data, especially when few labeled data are available, as can be seen in Table 4.

[1] F. Dan Foresee, M. Hagan, Gauss-newton approximation to bayesian learning, in: Neural Networks, 1997., International Conference on, Vol. 3, IEEE, 1997, pp. 1930–1935.

[2] D. MacKay, Bayesian interpolation, Neural computation 4 (3) (1992) 415–447.

[3] N. Treadgold, T. Gedeon, Exploring constructive cascade networks, Neural Networks, IEEE Transactions on 10 (6) (1999) 1335 –1350. doi:10.1109/72.809079.

[4] C. Bishop, Neural Networks for Pattern Recognition, 1st Edition, Oxford University Press, USA, 1996.

[5] Y. Jin, Neural network regularization and ensembling using multi-objective evolutionary algorithms, in: In: Congress on Evolutionary Computation (CEC'04), IEEE, IEEE Press, 2004, pp. 1–8.

[6] O. Ludwig, Study on non-parametric methods for fast pattern recognition with emphasis on neural networks and cascade classifiers, Ph.D. dissertation, University of Coimbra, Coimbra, Portugal, 2012.

[7] O. Ludwig, U. Nunes, Novel maximum-margin training algorithms for supervised neural networks, Neural Networks, IEEE Transactions on 21 (6) (2010) 972 –984.

[8] A. Deb, M. Gopal, S. Chandra, SVM-based tree-type neural networks as a critic in adaptive critic designs for control, Neural Networks, IEEE Transactions on 18 (4) (2007) 1016 –1030. doi:10.1109/TNN.2007.899255.

[9] S. Abe, Support Vector Machines for Pattern Classification (Advances in Pattern Recognition), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[10] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural networks 3 (5) (1990) 551–560.

[11] H. Demuth, M. Beale, Neural network toolbox user's guide, The MathWorks Inc.

[12] P. Bartlett, The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, Information Theory, IEEE Transactions on 44 (2) (1998) 525–536.

[13] R. Horn, C. Johnson, Matrix analysis, Cambridge university press, 1990.

[14] V. Vapnik, Statistical Learning Theory, John Wiley, 1998.

[15] M. Enzweiler, D. Gavrila, Monocular pedestrian detection: Survey and experiments, Pattern Analysis and Machine Intelligence, IEEE Transactions on 31 (12) (2009) 2179 –2195.

[16] O. Ludwig, U. Nunes, B. Ribeiro, C. Premebida, Improving the generalization capacity of cascade classifiers, Cybernetics, IEEE Transactions on PP (99) (2013) 1–12. doi:10.1109/TCYB.2013.2240678.

[17] S. Shalev-Shwartz, Y. Singer, N. Srebro, Pegasos: Primal estimated sub-gradient solver for svm, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 807–814.

[18] M. Adankon, M. Cheriet, Genetic algorithm–based training for semi-supervised svm, Neural computing & applications 19 (8) (2010) 1197–1206.

[19] D. Nguyen, B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, in: International Joint Conference on Neural Networks, IJCNN'90, IEEE, 1990, pp. 21–26.

[20] O. Ludwig, P. Gonzalez, A. Lima, Optimization of ANN applied to non-linear system identification, in: Proceedings of the 25th IASTED international conference on Modeling, indentification, and control, ACTA Press, Anaheim, CA, USA, 2006, pp. 402–407.

[21] O. Ludwig, U. Nunes, R. Araujo, L. Schnitman, H. Lepikson, Appli-

cations of information theory, genetic algorithms, and neural models to predict oil flow, Communications in Nonlinear Science and Numerical Simulation 14 (7) (2009) 2870 – 2885.

[22] O. Chapelle, B. Schölkopf, A. Zien, Semi-supervised learning, Vol. 2, MIT press Cambridge, MA, 2006.