Neural Networks 21 (2008) 1302–1310

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Data classification with multilayer perceptrons using a generalized error function

Luís M. Silva [a,*], J. Marques de Sá [a,b], Luís A. Alexandre [c,d]

[a] INEB-Instituto de Engenharia Biomédica, Porto, Portugal
[b] Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
[c] Dep. de Informática, Universidade da Beira Interior, Covilhã, Portugal
[d] IT, Networks and Multimedia Group, Covilhã, Portugal

ARTICLE INFO

Article history:
Received 9 November 2007
Received in revised form
31 March 2008
Accepted 28 April 2008

Keywords:
Multilayer perceptrons
Data classification
Error functions

ABSTRACT

The learning process of a multilayer perceptron requires the optimization of an error function $E(\mathbf{y}, \mathbf{t})$ comparing the predicted output, $\mathbf{y}$, and the observed target, $\mathbf{t}$. We review some usual error functions, analyze their mathematical properties for data classification purposes, and introduce a new one, E_{Exp}, inspired by the Z-EDM algorithm that we have recently proposed. An important property of E_{Exp} is its ability to emulate the behavior of other error functions by the sole adjustment of a real-valued parameter. In other words, E_{Exp} is a sort of generalized error function embodying complementary features of other functions. The experimental results show that the flexibility of the new, generalized, error function allows one to obtain the best results achievable with the other functions with a performance improvement in some cases.

1. Introduction

Multi-layer perceptrons (MLP) are a popular form of feedforward artificial neural networks with many successful applications in data classification. The supervised learning (training) process of an MLP with input data $\mathbf{x}$ and target $\mathbf{t}$, requires the use of an objective function (or error/cost/loss function) $E(\mathbf{y}, \mathbf{t})$ in order to assess the deviation of the predicted output values, $\mathbf{y} = \text{MLP}(\mathbf{x}; \mathbf{w})$ from the observed data values $\mathbf{t}$ and use that assessment for the convergence towards an optimal set of weights $\mathbf{w}^*$. There are many MLP training algorithms using the $\frac{\partial E}{\partial w}$ gradient information either directly or indirectly. In the present paper we concentrate on using the well-known backpropagation (BP) algorithm without loss of generalization of the main conclusions of the paper.

In what concerns the error function $E(\mathbf{y}, \mathbf{t})$, the well-known mean square error (MSE) function is by far the most commonly used, but as we will discuss later, it is not the most appropriate for data classification problems. There are other alternatives, such as the cross entropy (CE) error function and other entropy-based functions, which have been specifically applied by us to data classification problems (Santos, Alexandre, & Marques de Sá, 2004; Silva, Marques de Sá, & Alexandre, 2005). We have also proposed a new error function inspired on entropic criteria: the error density at the origin (Z-EDM) (Silva, Alexandre, & Marques de Sá, 2005, 2006). In this paper, we review these error functions, analyze their mathematical properties for data classification purposes and clarify aspects that are commonly omitted or have been misinterpreted in the literature. We also propose a new error function, E_{Exp}, inspired by the Z-EDM, which is capable of emulating the behavior of other error functions by the adjustment of a single real-valued parameter. Experimental results show that the flexibility of this new, generalized, exponential error function, E_{Exp}, allows one to achieve the best results achievable with the other functions, capitalizing on their complementary properties, with a performance improvement in some situations.

The paper is organized as follows: sections two to four present some error functions and study their behaviors in terms of the corresponding gradient; section five deals with monotonic error functions and presents the new error function; experimental results are reported in section six and finally the paper ends with some concluding remarks.

2. Usual error functions

We consider the usual classification problem where a pattern $\mathbf{x} \in \mathbb{R}^d$ is to be assigned to one of C classes, $(\omega_1, \ldots, \omega_C)$, by an MLP with one hidden layer and weight vector $\mathbf{w}$. The MLP is trained using a set of training pairs $(\mathbf{x}_i, \mathbf{t}_i)$, $i = 1, \ldots, N$, where each $\mathbf{t}_i = (t_{1,i}, \ldots, t_{C,i})$ is a realization of a (target) variable $\mathbf{t} = (t_1, \ldots, t_C)$ that describes the class to which $\mathbf{x}_i$ belongs in an *1-out-of-C* coding, with $t_k \in \{0, 1\}$ or $t_k \in \{-1, 1\}$ for $k = 1, \ldots, C$. Hence, the MLP

* Corresponding address: INEB - Instituto de Engenharia Biomedica, Campus da FEUPRua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal. Tel.: +351 22 508 16 23; fax: +351 22 508 16 24.
E-mail addresses: lmsilva@fe.up.pt (L.M. Silva), jmsa@fe.up.pt (J. Marques de Sá), lfbaa@di.ubi.pt (L.A. Alexandre).

0893-6080/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.04.004

has an output layer described by a vector $\mathbf{y} = (y_1, \ldots, y_C)$ that produces for each $\mathbf{x}_i$ its corresponding output $\mathbf{y}_i = (y_{1,i}, \ldots, y_{C,i})$. We assume this setting throughout the rest of the paper.

2.1. Mean square error

The MSE function is expressed as

$$E_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{t}_i - \mathbf{y}_i\|^2. \tag{1}$$

Originally derived for regression problems, the MSE function can be obtained by the maximum likelihood (ML) principle assuming the independence and Gaussianity of the target data (see Bishop (1995) for a detailed derivation of E_{MSE}).

Note, however, that the Gaussianity assumption of the target data in classification is not valid, due to its discrete nature (representing discrete class labels). Nevertheless, it can be shown (see below) that when using an *1-out-of-C* coding scheme for the targets, the MSE trained outputs of the network approximate the posterior probabilities of the class membership, $y_k = \hat{P}(\omega_k|\mathbf{x})$.

2.2. Cross-entropy

The CE cost function can also be derived from the maximum likelihood principle. Each component $y_k, k = 1, \ldots, C$ of the output vector is interpreted as an estimate of the posterior probability that input pattern $\mathbf{x}$ belongs to class ω_k, $y_k = \hat{P}(\omega_k|\mathbf{x})$ associated with a "true" distribution $\mathbf{p} = (p_1, \ldots, p_C)$ where $p_k = P(\omega_k|\mathbf{x}),\ k = 1, \ldots, C$.

Assuming that the classes are mutually exclusive, the true, $p(\mathbf{t}|\mathbf{x})$, and neural network, $p_{\mathbf{w}}(\mathbf{t}|\mathbf{x})$, probabilistic models for $\mathbf{t}$ can be described by the multinomial distributions:

$$p(\mathbf{t}|\mathbf{x}) = p_1^{t_1} p_2^{t_2} \ldots p_C^{t_C} \tag{2}$$

$$p_{\mathbf{w}}(\mathbf{t}|\mathbf{x}) = y_1^{t_1} y_2^{t_2} \ldots y_C^{t_C}. \tag{3}$$

We would like the model in (3) to approximate the true distribution (2). For that purpose we may apply the ML principle, or, equivalently, attempt to minimize the following Kulback-Leibler divergence which measures how well (3) approximates (2)

$$\sum_{i=1}^{N} \log\left(\frac{p(\mathbf{t}_i|\mathbf{x}_i)}{p_{\mathbf{w}}(\mathbf{t}_i|\mathbf{x}_i)}\right) = \sum_{i=1}^{N} \log\left(\frac{p_{1,i}^{t_{1,i}} \ldots p_{C,i}^{t_{C,i}}}{y_{1,i}^{t_{1,i}} \ldots y_{C,i}^{t_{C,i}}}\right)$$

$$= -\sum_{i=1}^{N}\sum_{k=1}^{C} t_{k,i} \log\left(y_{k,i}\right) + \sum_{i=1}^{N}\sum_{k=1}^{C} t_{k,i} \log\left(p_{k,i}\right). \tag{4}$$

Note that, as the values $p_{k,i} = P(\omega_k|\mathbf{x}_i)$ are unknown, (4) cannot be used as an error function. However, the $p_{k,i}$ do not depend on the parameters $\mathbf{w}$ of the MLP, which means that the minimization of (4) is equivalent to the minimization of

$$E_{CE} = -\sum_{i=1}^{N}\sum_{k=1}^{C} t_{k,i} \log\left(y_{k,i}\right). \tag{5}$$

Expression (5) is known in the literature as the *cross-entropy* error function. For the two-class case we only need one output such that $y = \hat{P}(\omega_1|\mathbf{x})$ $(1 - y = \hat{P}(\omega_2|\mathbf{x}))$ and the Bernoulli distribution is used for $p(t|\mathbf{x})$ and $p_{\mathbf{w}}(t|\mathbf{x})$. The designation *cross-entropy* associated to expression (5) may cause some confusion. In fact, the cross-entropy between two distributions p and q, which can be used as a measure of their discrepancy is defined as

$$-\sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}). \tag{6}$$

Its minimum value is obtained when $p = q$. Expression (5) is similar to (6) but one must note that the $t_{k,i}$, or more precisely $\mathbf{t}_i$, are *not* probabilities (are in fact random vectors with multinomial distribution). The role of $p(\mathbf{x})$ in (6) is played by the unknown $p(\mathbf{t}|\mathbf{x})$ as defined above. Since it is not dependent on the network's parameters, it is of no consequence for the minimization process, and thus, can be disregarded. Moreover, expression (6) is the distribution version of the cross-entropy, while (5) is the empirical one. There are some authors that interpret the $t_{k,i}$ as $P(\omega_k|\mathbf{x}_i)$. This is incorrect since as $t_{k,i} \in \{0, 1\}$ this would mean that a pattern would always be correctly classified. In fact, the $t_{k,i}$ in (5) are just acting as switches. When a particular $t_{k,i} = 1$ (which means that $\mathbf{x}_i$ belongs to class ω_k), then $y_{k,i}$ must be maximum and thus we just minimize $-\log(y_{k,i})$ (all the other $t_{j,i} = 0,\ j \neq k$).

2.3. Mean square error or cross entropy

From the above discussion it seems natural to choose the Kullback–Leibler or more precisely the cross-entropy error function to train neural network classifiers, because when interpreting the outputs as probabilities this is the optimal solution. In fact, the CE error function takes into account the binary characteristic of the targets. Several authors have studied the conditions that the outputs of a neural network must satisfy in order to use them as estimators of the posterior probabilities. In Gish (1990) and Richard and Lippmann (1991) it is shown that for an *1-out-of-C* coding scheme, with large N and a number of samples in each class that reflects the prior probabilities, networks trained with MSE provide outputs that are approximations of posterior probabilities. These authors also derived the CE cost function from the maximum likelihood or maximum mutual information principles and have arrived to the same conclusions. There are other reasons to choose CE. Several authors reported marked reductions on convergence rates and density of local minima (Matsuoka & Yi, 1991; Solla, Levin, & Fleisher, 1988) due to the characteristic steepness of the CE function. In fact, it is easy to see that slight changes on the output of the network have more effect when using CE than MSE, because cancelations in the error gradients generate high error gradients for outputs very distant from their targets. As a function of the absolute errors, MSE tends to produce large relative errors for small output values. As a function of the relative errors, CE is expected to estimate more accurately small probabilities (Baum & Wilczek, 1987; Bishop, 1995; Gish, 1990; Hinton, 1989; Solla et al., 1988).

3. Zero-error density maximization

Let us now define the error (deviation) variable $\mathbf{e} = \mathbf{t} - \mathbf{y}$. One can easily see that when using the *1-out-of-C* output coding, the errors regarding each class lie in disjoint hypercubes with the origin as their unique common point. The three-class case is represented in Fig. 1 assuming the following *1-out-of-C* coding: $\mathbf{t}(\omega_1) = (1, -1, -1)$, $\mathbf{t}(\omega_2) = (-1, 1, -1)$ and $\mathbf{t}(\omega_3) = (-1, -1, 1)$. Therefore, $\mathbf{e} = [-2, 2]^3$ with the errors in each class varying in a distinct hypercube. For instance, the ω_1 error corresponds to the hypercube $\mathbf{t}(\omega_1) - \mathbf{y} = (1, -1, -1) - (y_1, y_2, y_3) \in [0, 2] \times [-2, 0] \times [-2, 0]$, i.e., the rightmost shadowed hypercube in Fig. 1.

We would optimally like that during the training process of an MLP the output $\mathbf{y}$ gets as close as possible to the target $\mathbf{t}$ and thus the errors (deviations) convergent to the origin. For a given data set $\{(\mathbf{x}_i, \mathbf{t}_i)|i = 1, \ldots, N\}$, we would then get in this optimal scenario $\mathbf{e}_i = \mathbf{t}_i - \mathbf{y}_i = \mathbf{0}\ \forall i$, which amounts to a δ-Dirac distribution of the error variable centered at the origin. As a matter of fact, when using entropic error functions (Santos et al., 2004; Silva, Marques de Sá et al., 2005) we observe the tendency as training evolves of reaching higher peak distribution of the errors at the origin (the δ-

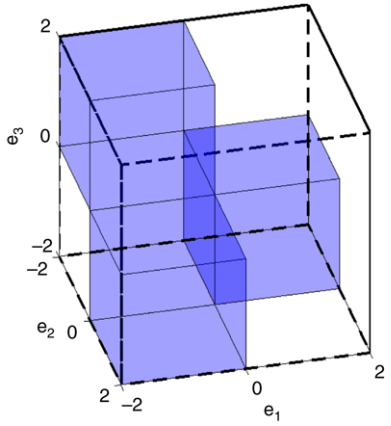


Fig. 1. Support space (shadowed cubes) for the error distribution in a three-class ($C = 3$) problem, $\mathbf{e} = (e_1, e_2, e_3)$.

Dirac distribution is the one with minimum entropy). This idea led us to adjust the weight vector $\mathbf{w}$ by maximizing the error density at the origin

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} f(\mathbf{0}; \mathbf{w}), \tag{7}$$

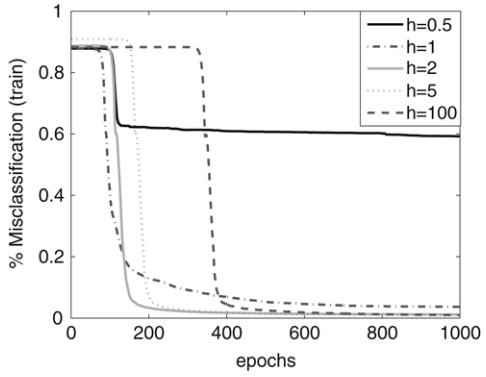
where $\mathbf{w}^*$ is the sought for optimal weight vector for the MLP and f is the error density (parameterized by $\mathbf{w}$). In practice, the error distribution is not known and making parametric assumptions would be very restrictive. Thus, we rely on nonparametric density estimation by using the well-known kernel density estimation procedure of Parzen windows (Silverman, 1986). Given a set of errors $\mathbf{e}_1, \ldots, \mathbf{e}_N$, the estimated density at $\mathbf{e} = \mathbf{0}$ is

$$\hat{f}(\mathbf{0}) = \frac{1}{Nh^C} \sum_{i=1}^{N} K\left(\frac{\mathbf{0} - \mathbf{e}_i}{h}\right), \tag{8}$$

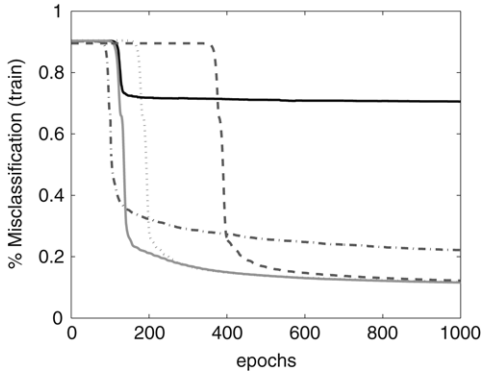
where K is a multidimensional kernel function, h is the smoothing parameter (kernel bandwidth) and C the dimension of $\mathbf{e}$ (number of classes). The multivariate Gaussian kernel with zero mean and unit covariance (Silverman, 1986) is chosen for K. This is because of its continuity and differentiability, which are crucial properties for the BP algorithm. Also, as proven in Silva et al. (2006), the Gaussian kernel satisfies the conditions needed to ensuring that the use of kernel density estimation does not affect the optimal solution. The final expression to be maximized becomes:

$$\hat{f}(\mathbf{0}) = \frac{1}{Nh^C} \sum_{i=1}^{N} \frac{1}{(2\pi)^{C/2}} \exp\left(-\frac{\mathbf{e}_i^2}{2h^2}\right). \tag{9}$$

This procedure, named Zero-Error Density Maximization (Z-EDM), inspired by our previous work on using entropic error measures, can be easily plugged in the usual backpropagation scheme as described in Silva, Alexandre et al. (2005) and Silva et al. (2006). Note that expression (9) depends on the kernel bandwidth h. This parameter controls the smoothness of the density estimate and consequently the smoothness of the error function. In order to better understand the influence of h in the training process several data sets were used for training 100 times (full data set) using several different values of h. Fig. 2 shows the mean training curves (the standard deviations are small) for two data sets: Olive and Ctg16. We found that a value of h smaller than 1 does not work, independently of the data set, number of classes and/or the number of training examples. When h is increased, the curve is basically shifted forward and a flat region appears in the earlier epochs. This general behavior was found to be the same for all tested data sets, regardless of the number of classes and number of examples available for training.



(a) Olive: 9 classes, 8 features and 572 patterns.



(b) Ctg16: 10 classes, 16 features and 2126 patterns.

Fig. 2. Mean training curves with Z-EDM for different values of h in two data sets.

Let us now consider, for simplicity's sake, an MLP with one output for a two-class problem. The gradient of (9) with respect to a particular parameter w (weight of the MLP) is derived as

$$\frac{\partial \hat{f}(0)}{\partial w} = -\frac{1}{Nh} \sum_{i=1}^{N} \frac{e_i}{h^2\sqrt{2\pi}} \left[\exp\left(-\frac{e_i^2}{2h^2}\right)\right] \frac{\partial e_i}{\partial w}. \tag{10}$$

In this expression we may consider the function

$$\varphi(e) = \frac{e}{Nh^3\sqrt{2\pi}} \exp\left(-\frac{e^2}{2h^2}\right) \quad e \in [-2, 2] \tag{11}$$

as a weight function of the gradient "particle" $\frac{\partial e}{\partial w}$. Fig. 3 shows $\varphi(e)$ for some values of N and h. We can see that gradient particles corresponding to larger absolute values of e get larger weights, whereas gradient particles corresponding to smaller values of e will have a small contribution to the update value (10) of the parameter w. Of course, if we increase N or h, then $\varphi(e) \to 0$ and this is the reason for the initial flat platforms encountered in the first epochs of the training error. If we also look to the order of magnitude of the values given by (11), we can conclude that this behavior is due to the initial convergence "effort" being done by the adaptive learning rate procedure[1] while attempting to compensate those minuscule orders of magnitude.

In fact, we can make some modifications to our error function in order to avoid this problem. Note that the maximization of (9) is equivalent to the maximization of

$$E_{ZEDM} = \sum_{i=1}^{N} h^2 \exp\left(-\frac{\mathbf{e}_i^2}{2h^2}\right) \tag{12}$$

[1] We use an adaptive learning rate procedure as described in Silva, Alexandre et al. (2005).

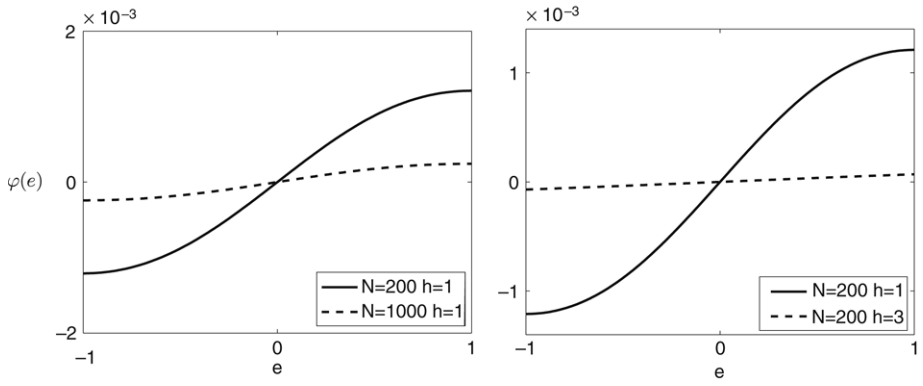


Fig. 3. $\varphi(e)$ as in (11) for different values of N and h. For better visualization we restricted e to the interval $[-1,\ 1]$.

in the sense that the same solutions are encountered, because $\frac{1}{Nh^C(2\pi)^{C/2}}$ and h^2 are just positive scaling factors (we use the factor h^2 to allow a simplification of the gradient).

4. Gradient analysis

Let us now compare the gradients of MSE, CE and Z-EDM. For simplicity we analyze the one output case (two-class problem). We have

$$\frac{\partial E_{\text{MSE}}}{\partial w} = -\sum_{i=1}^{N} e_i \frac{\partial y_i}{\partial w}, \tag{13}$$

$$\frac{\partial E_{\text{CE}}}{\partial w} = -\sum_{i=1}^{N} \frac{e_i}{y_i(1-y_i)} \frac{\partial y_i}{\partial w}, \tag{14}$$

$$\frac{\partial E_{\text{ZEDM}}}{\partial w} = \sum_{i=1}^{N} \exp\left(-\frac{e_i^2}{2h^2}\right) e_i \frac{\partial y_i}{\partial w}. \tag{15}$$

Note that we now write the gradients in terms of the gradient particle $\frac{\partial y}{\partial w}$. Consider the following weight functions

$$\varphi_{\text{MSE}}(e) = e, \tag{16}$$

$$\varphi_{\text{CE}}(y) = \frac{e}{y(1-y)} = \frac{t-y}{y(1-y)} \qquad t \in \{0, 1\}, \tag{17}$$

$$\varphi_{\text{ZEDM}}(e) = \exp\left(-\frac{e^2}{2h^2}\right) e. \tag{18}$$

Fig. 4 shows these functions for some values of their corresponding parameters. From this figure we can see the linear behavior of φ_{MSE}: the gradient particles $\frac{\partial y_i}{\partial w}$ have a weight equal to the corresponding error. The φ_{CE} function also confers larger weights to gradient particles corresponding to larger errors. Note that, when $t = 1$ ($t = 0$) larger errors correspond to y closest to zero (one). However, the weight attribution is not linear but hyperbolic. For $\varphi_{\text{Z-EDM}}$ we can distinguish three basic behaviors. When we let $h \to 0$ then $\varphi_{\text{Z-EDM}} \to 0$. If the parameters of the network are initialized in $[-b, b]$ with b close to zero, then in the early phase of the training process, all the errors are around the values -1 and 1. Thus, with h too small, the algorithm will have difficulties to converge (or even will not be able to start at all!) because $\varphi_{\text{Z-EDM}}$ gives weights close to zero. For moderate h ($h \approx 2$), $\varphi_{\text{Z-EDM}}$ is a nonlinear function (similar to a sigmoid) where, again, gradient particles corresponding to larger errors get larger weights. Note, however, the contrast with φ_{CE}: for larger errors φ_{CE} "accelerates" the weight value while $\varphi_{\text{Z-EDM}}$ "decelerates". Finally, when h is large, $\varphi_{\text{Z-EDM}}$ behaves like φ_{MSE}. In fact, it is easy to see that $\lim_{h\to+\infty} \varphi_{\text{Z-EDM}} = \varphi_{\text{MSE}}$.

5. Monotonic error functions

As discussed earlier, when a neural network is trained using MSE or CE minimization, its outputs approximate the posterior probabilities of class membership. Thus, in the presence of large data sets, it tends to produce optimal solutions in the Bayes sense. However, as argued in Hampshire and Waibel (1990) and Møller (1993), minimization of the error function does not necessarily imply misclassification minimization in practice (especially for small data sets or in the presence of local minima). Sub-optimal solutions may occur due to flat regions in weight space. This can be seen with a simple example. Let us assume a two class problem with one output *per* class. The squared error for a particular pattern $\mathbf{x}$ from class ω_1 can be written as (considering $\mathbf{t}(\omega_1) = (1, 0)$)

$$E(\mathbf{x}) = (t_1 - y_1)^2 + (t_2 - y_2)^2 \tag{19}$$

$$= (1 - y_1)^2 + y_2^2. \tag{20}$$

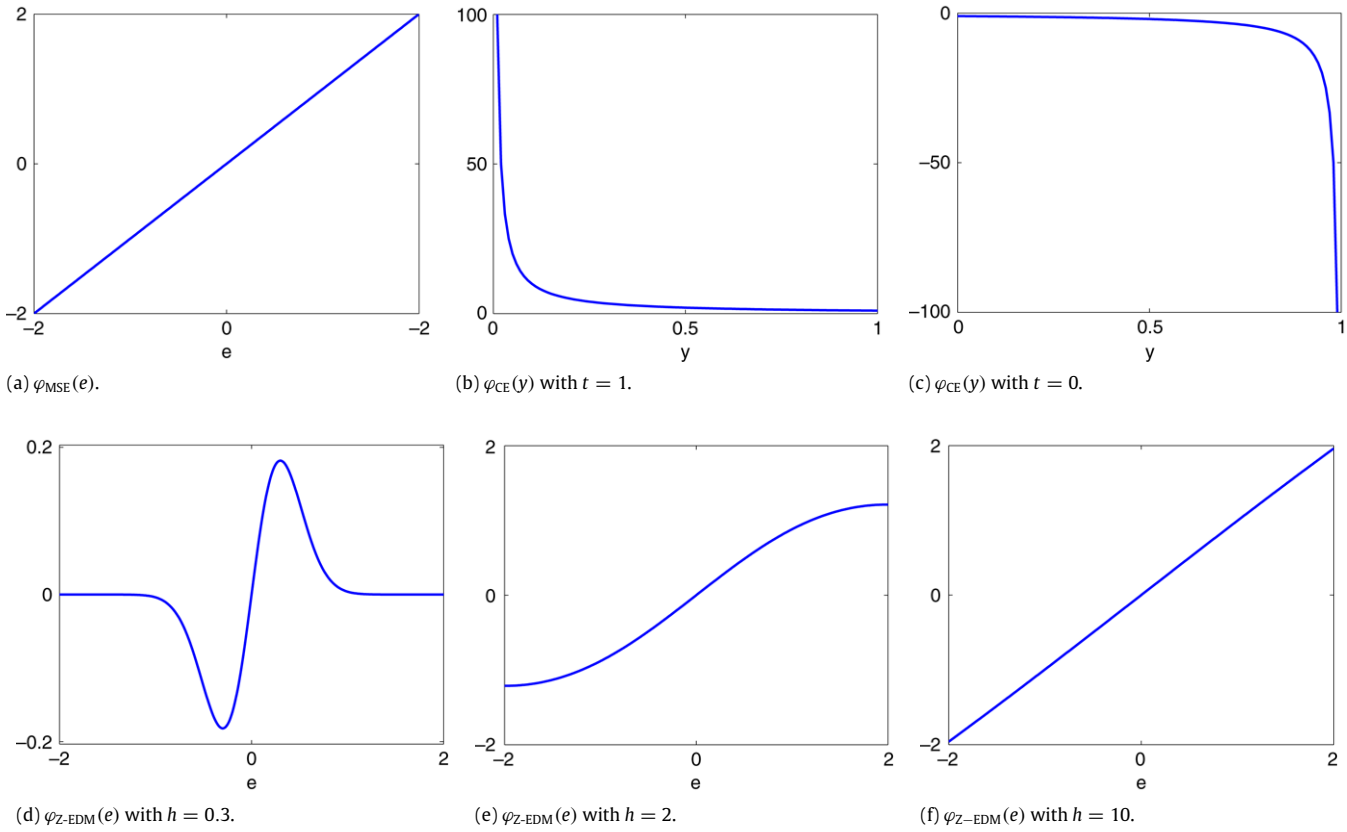
The contours of E are shown in Fig. 5. Using the rule that $\mathbf{x}$ belongs to class ω_1 if $y_1(\mathbf{x}) > y_2(\mathbf{x})$ we can see that while $\mathbf{x}_2$ is correctly classified, $\mathbf{x}_1$ is misclassified. However, $\mathbf{x}_1$ has a *lower* MSE than $\mathbf{x}_2$, which reinforces the idea that sub-optimal solutions may occur. The same happens with CE. Thus, minimization of these error functions does not imply misclassification minimization. For this reason, they were designated *non-monotonic* in Hampshire and Waibel (1990). In the same work, a monotonic error function is proposed: classification figures of merit (CFM_{mono}). This error function focuses mostly on the reduction of misclassification (this is known as *differential learning*) and not on achieving an exact convergence to the target values. However, training with CFM_{mono} was found to be much slower than with MSE or CE.

5.1. A simple monotonic error function

We can define a simple monotonic error function for a two-class problem. We consider an MLP with one output *per* class $\mathbf{y} = (y_1, y_2)$ and a class encoding defined by $\mathbf{t} = (t_1, t_2) = (1, -1)$ and $\mathbf{t} = (t_1, t_2) = (-1, 1)$ for classes ω_1 and ω_2, respectively. By the definition presented above, a monotonic error function should have contours parallel to $y_1 = y_2$. This can be achieved with the following error function:

$$E_{\text{SMF}} = \frac{1}{2} \sum_{i=1}^{N} [(y_{1i} - y_{2i}) - (t_{1i} - t_{2i})]^2. \tag{21}$$

This is a simple transformation (rotation and shifting) of the parabolic cylinder $z = x^2$. Note that as $E_{\text{SMF}} \geq 0$ it has a global minimum of $E = 0$ when the outputs equal the targets. Fig. 6 shows the error surface and corresponding contour plot for patterns from ω_1 and ω_2.



(a) $\varphi_{\text{MSE}}(e)$. (b) $\varphi_{\text{CE}}(y)$ with $t = 1$. (c) $\varphi_{\text{CE}}(y)$ with $t = 0$.

(d) $\varphi_{\text{Z-EDM}}(e)$ with $h = 0.3$. (e) $\varphi_{\text{Z-EDM}}(e)$ with $h = 2$. (f) $\varphi_{\text{Z-EDM}}(e)$ with $h = 10$.

Fig. 4. Gradient-weighting functions for MSE, CE and Z-EDM.

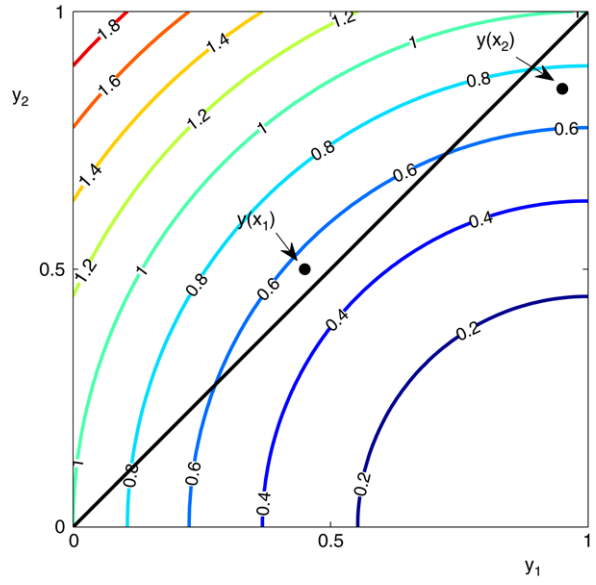


Fig. 5. Contours of MSE for a ω_1 pattern.

The gradient can be calculated as

$$\left(\frac{\partial E_{\text{SMF}}}{\partial y_{1i}}, \frac{\partial E_{\text{SMF}}}{\partial y_{2i}}\right) = ((y_{1i} - y_{2i}) - (t_{1i} - t_{2i}), \\ -[(y_{1i} - y_{2i}) - (t_{1i} - t_{2i})]), \quad (22)$$

where we note that $\frac{\partial E}{\partial y_{1i}} = -\frac{\partial E}{\partial y_{2i}}$.

The flexibility of E could be enhanced using the following version of (21):

$$E_{\text{SMF}} = \frac{1}{2}\sum_{i=1}^{N}[(y_{1i} - y_{2i}) - (t_{1i} - t_{2i})]^{\gamma}. \quad (23)$$

The parameter γ is an even integer positive number ensuring that E_{SMF} is always non-negative. The major effect of increasing γ is exemplified in Fig. 7. The steepness is increased in regions far from the desired target, whereas in regions near the desired target the function is flattened.

5.2. Exponential error function

Møller (1993) proposed an error function with a *soft-monotonicity* property, controlled by a positive parameter. It is defined as

$$E_{\text{Moller}} = \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{C}\exp\left(-\alpha(y_{k,i} - t_{k,i} + \beta)(t_{k,i} + \beta - y_{k,i})\right) \quad (24)$$

and was designated *exponential error function*. The parameters β and α are positive; β is the width of a region, $\mathcal{R}$, of acceptable error around the desired target and α controls the steepness of the error function outside that region ($\bar{\mathcal{R}}$). If we increase α then E_{Moller} becomes more steep in $\bar{\mathcal{R}}$, forcing the outputs towards the boundary of $\mathcal{R}$. By decreasing β, the outputs are pulled towards the desired targets (see Møller (1993) for a detailed discussion). When $\alpha \to +\infty$, E_{Moller} becomes monotonic, while varying α the degree of monotonicity is controled (soft-monotonicity).

5.3. Another exponential error function

The analysis of MSE, CE and Z-EDM gradient behaviors (presented in Section 4), suggested that it might be possible to

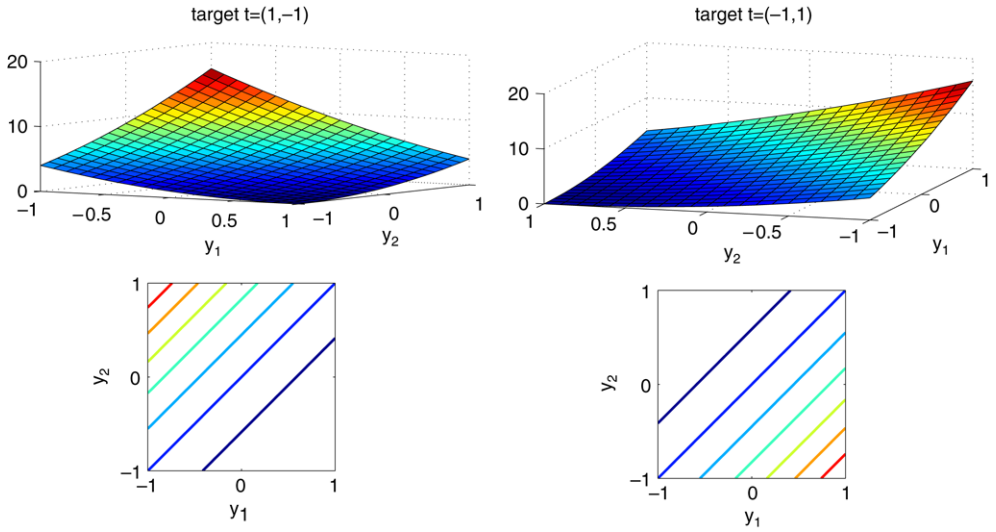


Fig. 6. *Left*: Error surface and contour plots of E_{SMF} in the presence of a pattern from ω_1; *Right*: The same but for a pattern from ω_2.

Fig. 7. Effect of increasing the power of a polynomial function x^{γ}.

create a parameterized error function capable of "emulating" those gradients. This error function (another exponential error function) is a generalization of (12), that allows positive arguments in the exponential function. It is expressed as

$$E_{\text{Exp}} = \sum_{i=1}^{N} \tau \exp\left(\mathbf{e}_i^2/\tau\right) = \sum_{i=1}^{N} \tau \exp\left(\frac{1}{\tau}\sum_{k=1}^{C} e_{k,i}^2\right), \tag{25}$$

where τ is a real number. Clearly, E_{Exp} resembles E_{Moller} for $\beta = 0$. There is, however, a significant difference: the location of the sum over k. In detail, in E_{Moller} we sum the exponentials of the (squares of) errors while in E_{Exp} we compute the exponential of the sum of those quantities. This implies a fundamental difference in terms of the gradients (for $\beta = 0$):

$$\frac{\partial E_{\text{Exp}}}{\partial y_{k,i}} = -2\sum_{i}\left[\exp\left(\frac{1}{\tau}\sum_{k=1}^{C} e_{k,i}^2\right)\right] e_{k,i} \tag{26}$$

$$\frac{\partial E_{\text{Moller}}}{\partial y_{k,i}} = -\sum_{i} \alpha\left[\exp\left(+\alpha e_{k,i}^2\right)\right] e_{k,i}. \tag{27}$$

We see that with E_{Exp}, the backpropagated error through the output y_k uses information from *all* other outputs (present in $\exp\left(\frac{1}{\tau}\sum_{k=1}^{C} e_{k,i}^2\right)$), while E_{Moller} only uses the error associated to that particular output (present in $\exp\left(+\alpha e_{k,i}^2\right)$). It is easy to see

Fig. 8. Plot of the gradient of E_{Exp} for one output and different positive values of τ. For visualization purposes, e is restricted to the interval $[-1,\ 1]$.

that if $\tau < 0$, E_{Exp} recovers the (negative) Z-EDM error function. Thus, we may also say that for $\tau \to -\infty$, E_{Exp} behaves like MSE. When $\tau > 0$, E_{Exp} behaves like CE. This can be seen in Fig. 8 where the gradient of E_{Exp} (for one output) is plotted for different positive values of τ. From small to moderate values of τ, the function has a marked hyperbolic shape, in the same sense as CE: smaller errors get smaller weights with an "accelerated" increasing when the errors get larger. The parameter τ is, as α in E_{Moller}, controlling the steepness of E_{Exp}. Also for $\tau \to \infty$, E_{Exp} behaves like MSE. In summary, E_{Exp} has the appropriate *flexibility to emulate the whole range of Z-EDM - MSE - CE behaviors with a control of the degree of monotonicity*.

6. Experiments

To evaluate and compare the capabilities of the different error functions presented, we conducted an experimental procedure using several publicly available data sets (mainly from the UCI repository (Blake & Merz, 1998)). The problems were chosen so as to comprise a wide range of real world applications as well as to include several complexities: different number of features, classes and patterns. Table 1 summarizes the main characteristics of these data sets.