Pattern Recognition 47 (2014) 441–453

Contents lists available at ScienceDirect

Pattern Recognition

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

The C-loss function for pattern classification

Abhishek Singh [a,*], Rosha Pokharel [b], Jose Principe [b]

[a] Department of Electrical & Computer Engineering, University of Illinois at Urbana-Champaign, United States
[b] Department of Electrical & Computer Engineering, University of Florida, Gainesville, United States

ARTICLE INFO

Article history:
Received 3 October 2012
Received in revised form
16 July 2013
Accepted 24 July 2013
Available online 3 August 2013

Keywords:
Classification
Correntropy
Neural network
Loss function
Backprojection

ABSTRACT

This paper presents a new loss function for neural network classification, inspired by the recently proposed similarity measure called Correntropy. We show that this function essentially behaves like the conventional square loss for samples that are well within the decision boundary and have small errors, and L_0 or counting norm for samples that are outliers or are difficult to classify. Depending on the value of the kernel size parameter, the proposed loss function moves smoothly from convex to non-convex and becomes a close approximation to the misclassification loss (ideal 0–1 loss). We show that the discriminant function obtained by optimizing the proposed loss function in the neighborhood of the ideal 0–1 loss function to train a neural network is immune to overfitting, more robust to outliers, and has consistent and better generalization performance as compared to other commonly used loss functions, even after prolonged training. The results also show that it is a close competitor to the SVM. Since the proposed method is compatible with simple gradient based online learning, it is a practical way of improving the performance of neural network classifiers.

1. Introduction

Classification aims at assigning class labels to data using an 'optimal' decision rule that is learnt using a set of pre-labeled training samples. This 'optimal' decision rule or discriminant function f is learnt by minimizing the empirical risk, which is a sample average of a loss function. The loss (function of the prediction $f(\mathbf{x})$, and the true label y) is essentially the price we pay for predicting the label to be $f(\mathbf{x})$, instead of y. This procedure for learning the discriminant function is called the Empirical Risk Minimization, and is a widely used principle for classification and statistical learning [1,2].

The most natural loss function for classification is the misclassification error rate (or the 0–1 loss)

$$l_{0-1}(f(\mathbf{x}), y) = \| (-yf(\mathbf{x}))_+ \|_0, \tag{1}$$

where $(.)_+$ denotes the positive part and $\|.\|_0$ denotes the L_0 norm. This essentially is a count of the number of incorrect classifications made by the decision rule f. Therefore, the 0–1 loss function directly relates to the probability of misclassification. Optimization of the risk based on such a loss function, however, is computationally intractable due to its non-continuity and non-convexity [1,2]. Therefore, a surrogate loss function is applied to many classification procedures. For example, well known loss functions for training the weights of a neural network or a radial basis function (RBF) network are the squared loss, $(y-f(\mathbf{x}))^2$, or $(1-yf(\mathbf{x}))^2$, and the logistic loss, $\log(1+e^{-yf(\mathbf{x})})$. The Support Vector Machine (SVM) [3,4] uses the hinge loss, $[1-yf(\mathbf{x})]_+$.

Within the statistical learning community, convex surrogates of the 0–1 misclassification loss are highly preferred because of the virtues that convexity brings – unique optima, efficient optimization using convex optimization tools and amenability to theoretical analysis of error bounds [5]. However, convex functions are still poor approximations to the 0–1 loss function. They tend to be boundless and offer poor robustness to outliers [2]. Another important limitation is that the complexities of convex optimization algorithms grow very fast with more data [6]. Some non-convex loss functions have been proposed recently with the aim of addressing these issues [7,8].

There is a large class of problems where optimization cannot be done using convex programming techniques. For example, training of deep networks for large scale AI problems primarily rely on online, gradient-based methods [9,10]. Such neural network based learning machines can benefit from non-convex loss functions, as they can potentially offer better scalability, robustness and generalization performance. Although non-convex optimization and loss functions do not offer many theoretical guarantees, the empirical evidence that they work better in engineering applications is becoming overwhelming [6].

A loss function for classification that is inspired by the statistical measure called Correntropy [11] was proposed in [12]. Correntropy

* Corresponding author. Tel.: +1 3526154195.
E-mail addresses: abhishek486@gmail.com, asingh18@illinois.edu, abhishek_singh@ieee.org (A. Singh), rosha@cnel.ufl.edu (R. Pokharel), principe@cnel.ufl.edu (J. Principe).

0031-3203/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2013.07.017

between two random variables is a generalized correlation function or a robust measure of statistical similarity, which makes use of second and higher order statistics. It has been successfully applied to problems like robust regression [13], adaptive filtering [14–17], pitch detection in speech [18,19], MACE (Minimum Average Correlation Energy) filtering for object recognition [20], etc. In a classification setting, maximizing the similarity between the prediction $f(\mathbf{x})$ and the target y in the Correntropy sense, effectively induces a non-convex, smooth loss function (which we call C-loss) that can be used to train a classifier using an online gradient based technique.

This paper extends our earlier work in [12] and further characterizes the C-loss function for classification. We examine the performance of a single hidden layer perceptron trained with the C-loss function (using backpropagation) over different system parameters such as training epochs and network size. We obtain better generalization results on several synthetic and real world datasets, when compared to the traditional squared loss function.

Furthermore, we demonstrate the performance of the C-loss function while training RBF networks as well. We obtain superior results using the C-loss function when compared to the logistic loss function, while training an RBF classifier.

We also compare the performance of the proposed loss function with SVMs, that use the hinge loss function.

Conventional neural network based classifiers suffer from the problem of overfitting due to overtraining, which often causes poor generalization. In all the abovementioned experiments, we show that classifiers trained using the proposed C-loss function are more robust to overfitting even on prolonged training, and are able to maintain consistent generalization performance. The proposed online method of training classifiers using the C-loss function has the overall practical appeal that it offers more consistent and better generalization performance at no additional computational cost.

The next section formalizes the pattern classification problem from the point of view of statistical learning. Section 3 introduces the C-loss function, along with some of its properties. In Section 4 we discuss how the C-loss function can be used to train neural network based classifiers. Section 5 presents our experimental results. We compare the performance of the C-loss function to the square loss (on MLPs), the logistic loss (on RBF networks), and SVMs on several real world datasets obtained from UCI Machine Learning Repository [21], using different neural network architectures, and several different system parameters. In Sections 6 and 7 we present some important discussions and insights and draw conclusions.

2. Statistical theory of classification

2.1. Loss functions and risk

Suppose we are given a training set of observations $D_n = \{(\mathbf{x}_i,\ y_i),\ i=1,2,\ldots,n\}$, assumed to be i.i.d. realizations of a random pair $(\mathbf{X},Y)$. Here, $\mathbf{X}\in\mathcal{X}$ is the input vector and $Y\in\{-1,1\}$ is the class label (we consider a binary classification problem for now). The goal of classification is to select a function f from a class of functions $\mathcal{F}$, such that the sign of $f(\mathbf{X})$ is an accurate prediction of Y under an unknown joint distribution $P(\mathbf{X},Y)$. In other words, we want to select $f\in\mathcal{F}$ that minimizes the risk $R(f)$ given by

$$R(f) = E[l_{0-1}(Yf(\mathbf{X}))] = P(Y \neq \text{sign}(f(\mathbf{X}))). \tag{2}$$

The product $yf(\mathbf{x})$ is called the margin (denoted by α) and can be treated as a measure of correctness of the decision for the sample $\mathbf{x}$. Given a sample set D_n of realizations, it is natural to consider the empirical risk, or the sample average of the 0–1 loss

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} l_{0-1}(y_i f(\mathbf{x}_i)). \tag{3}$$

Optimization of the empirical risk as above, however, is computationally intractable, primarily because of the discontinuity of the 0–1 loss function. The optimization procedure therefore involves choosing a surrogate $\phi(\alpha)=\phi(yf(\mathbf{x}))$ as the loss function. The result is the minimization of the $\phi-$risk and empirical $\phi-$risk defined by the following:

$$R_\phi(f) = E[\phi(Yf(\mathbf{X}))] \tag{4}$$

$$\hat{R}_\phi(f) = \frac{1}{n}\sum_{i=1}^{n} \phi(y_i f(\mathbf{x}_i)). \tag{5}$$

Fig. 1 shows three commonly used surrogate loss functions – the hinge loss used in SVMs, the square loss and the logistic loss used in training neural networks and RBF networks.

In addition to making the optimization of the risk tractable, choosing a surrogate loss function has another motivation. Minimizing the sample average of an appropriately well behaved loss function may have a regularizing effect [22].

2.2. Bayes' Optimal decision rule

Let $p(\mathbf{x}) = P(Y=1|\mathbf{X}=\mathbf{x})$ be the conditional probability of the positive class given $\mathbf{X}=\mathbf{x}$. Then, the decision-theoretic optimal classification rule with the smallest generalization error is sign$[p(\mathbf{x})-1/2]$. This is called the Bayes' optimal rule. The risk associated with the Bayes' optimal rule is called the Bayes' optimal risk $R^*=R(f^*)$.

2.3. Fisher consistency

Definition 1. A margin-based loss function $\phi(yf(\mathbf{x}))$ is said to be Fisher consistent or 'classification calibrated' if the population minimizer f^* of the expected risk $E[\phi(Yf(\mathbf{X}))]$ has the same sign as the Bayes' optimal decision rule sign$[p(\mathbf{x})-1/2]$.

Fisher consistency simply provides the reassurance that optimizing a surrogate loss does not ultimately hinder the search for a discriminant function that achieves the Bayes' optimal risk. Lin [23] states a theorem that can be used to easily check if a given function is Fisher consistent.

Theorem 1. *If V is a function satisfying the following two assumptions*:

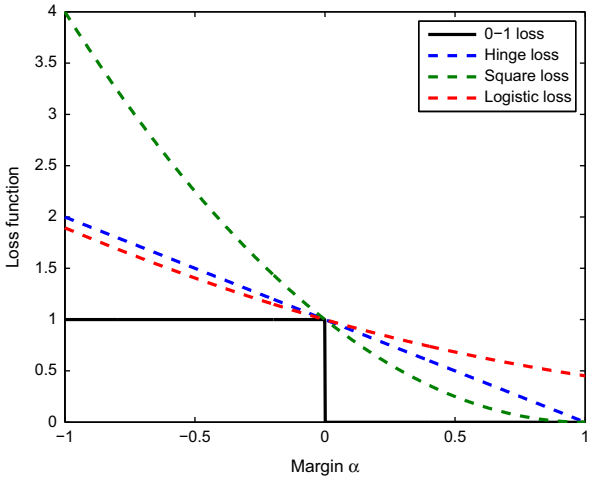

Fig. 1. The hinge and square loss functions, plotted along with the 0–1 loss.

1. $V(z) < V(-z), \quad \forall z > 0$
2. $V'(0) \neq 0$ *exists*

then, if $E[V(Yf(\mathbf{X}))]$ *has a global minimizer* $f^*(\mathbf{x})$, *then* $sign[f^*(\mathbf{x})] = sign[p(\mathbf{x}) - 1/2]$.

The proof is fairly straightforward and can be found in [23].

3. Loss function induced by Correntropy

Cross Correntropy or simply Correntropy between two random variables X and Y is a generalized similarity measure defined as

$$v(X,Y) = E[\kappa_\sigma(X-Y)], \tag{6}$$

where $\kappa_\sigma(z)$ is defined to be a radial basis (Gaussian) function with width parameter σ, centered at zero and evaluated at z. In practice, given only a finite number of realizations of the random variables, Correntropy between them is computed as

$$\hat{v}(X,Y) = \frac{1}{n}\sum_{i=1}^{n} \kappa_\sigma(x_i - y_i). \tag{7}$$

Correntropy is a measure of how similar two random variables are, within a small neighborhood determined by the kernel width σ. On the other hand, metrics such as mean squared error (MSE) provide a global measure. The localization provided by the kernel width proves to be very useful in reducing the detrimental effects of outliers and impulsive noise. A measure based on just second order statistics such as MSE, can easily get biased in such conditions.

In a classification setting, the goal is to maximize the similarity between the classifier output and the true label, in the Correntropy sense. Therefore, the loss function should be chosen such that minimization of the expected risk is equivalent to maximization of Correntropy. We therefore define the Correntropy induced loss function or the C-loss function as

$$l_C(y, f(\mathbf{x})) = \beta[1 - \kappa_\sigma(y - f(\mathbf{x}))]. \tag{8}$$

This can also be expressed in terms of the classification margin[1] $\alpha = yf(\mathbf{x})$ as

$$l_C(\alpha) = \beta[1 - \kappa_\sigma(1-\alpha)] \tag{9}$$

$$l_C(\alpha) = \beta[1 - \kappa_\sigma(1 - yf(\mathbf{x}))]. \tag{10}$$

β is a positive scaling constant chosen such that $l_C(\alpha = 0) = 1$. Therefore $\beta = [1 - \exp(-1/2\sigma^2)]^{-1}$. Fig. 2 shows plots of the C-loss function, for different values of the kernel width parameter σ, plotted against the margin α. The 0–1 loss is also shown. The C-loss function has the distinct advantage that the kernel size σ (or bandwidth) tunes the behavior of the C-loss function from the square loss ($\sigma = 2$, and above) to the hinge loss ($\sigma = 1$), and, for smaller values, approaches the 0–1 loss function.

The expected risk associated with the C-loss function is

$$R_C(f) = \beta(1 - E[\kappa_\sigma(1 - yf(\mathbf{x}))]) \tag{11}$$

$$R_C(f) = \beta(1 - E[\kappa_\sigma(y - f(\mathbf{x}))]) \tag{12}$$

$$R_C(f) = \beta(1 - v(Y, f(\mathbf{X}))). \tag{13}$$

Clearly, minimizing the above risk is equivalent to maximizing the similarity (in the Correntropy sense) between the predicted label $f(x)$ and the true label y.

[1] Both $\alpha = yf(\mathbf{x})$, and $e = y - f(\mathbf{x})$ are quantities that measure the 'margin' or the distance of the sample $\mathbf{x}$ from the discriminant function. We refer to either or both these quantities throughout the paper.

If we make the change of variables, $\mathcal{E} = Y - f(\mathbf{X})$, the estimator of Correntropy can be written as

$$\hat{v}(\mathcal{E}) = \frac{1}{n}\sum_{i=1}^{n} \kappa_\sigma(e_i). \tag{14}$$

From the Parzen density estimation principle [24], it can be easily seen that the above quantity is simply an estimator of the *pdf* of $\mathcal{E}$, evaluated at 0. Therefore, maximizing the Correntropy of the errors $\mathcal{E}$ of a classifier, with small kernel sizes, essentially maximizes $p(\mathcal{E} = 0)$. This is appropriate for regression problems as demonstrated in [14,25]. But in a classification setting, the dynamic range of the error is predefined *a priori*. Therefore, the kernel size should be set such that the C-loss function approximates the 0–1 loss function.

We now discuss some relevant properties of the C-loss function that motivate its use as a robust loss for training classifiers. For an exhaustive coverage of all properties of Correntropy, the reader is referred to [11,13].

Property 1. *The C-loss function* $l_C(\alpha)$ *is Fisher consistent (or 'classification calibrated') for all positive values of the kernel width* σ, *and the population minimizer* f^* *of the expected risk* $E[l_C(Yf(\mathbf{X}))]$ *yields the Bayes' optimal decision rule.*

Proof.

$$l_C(\alpha) = \beta\left[1 - \exp\left(\frac{-(1-\alpha)^2}{2\sigma^2}\right)\right], \tag{15}$$

It is easy to observe the following:

1. $l_C(\alpha) < l_C(-\alpha), \quad \forall \alpha > 0$, and,
2. $l_C'(\alpha) = -\beta \exp\left(\frac{-(1-\alpha)^2}{2\sigma^2}\right)\left(\frac{1-\alpha}{\sigma^2}\right)$ (16)

   Therefore, $l_C'(0) \neq 0$ exists.

Using the above results and Theorem 1, we infer that if $f^*(\mathbf{x})$ is the global minimizer of $R_C(f)$, then $\text{sign}[f^*(\mathbf{x})] = \text{sign}[p(\mathbf{x}) - 1/2]$. Therefore, the loss function $l_C(\alpha)$ is Fisher consistent or classification calibrated. This property holds for all positive values of the kernel width parameter σ.

Qualitatively, Fisher consistency ensures that a higher price is paid for incorrect classification of samples as compared to correctly classifying them. □

Property 2. *The expected risk obtained using the C-loss is a function of second and higher order moments of the margin.*

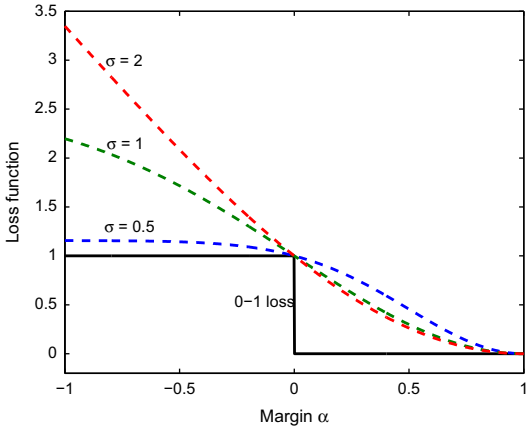

Fig. 2. The C-loss function for different values of the kernel width parameter σ, plotted along with the 0–1 loss.

Proof. Using Taylor's series expansion of the exponential function, the expected risk in (11) can be expanded as

$$R_C(f)=\beta\left[1-\sum_{k=0}^{\infty}\frac{(-1)^k}{(2\sigma^2)^k k!}E\left[(1-Yf(\mathbf{X}))^{2k}\right]\right] \tag{17}$$

$$R_C(f)=\beta\left[1-\sum_{k=0}^{\infty}\frac{(-1)^k}{(2\sigma^2)^k k!}E\left[(Y-f(\mathbf{X}))^{2k}\right]\right] \tag{18}$$

The risk is therefore a function of all even moments of difference in the prediction $f(\mathbf{X})$ and the true label Y. The risk computed using the C-loss includes information from higher order statistics of the error or the margin. On the other hand, using the squared loss only includes the second order moment or the variance, which is just one term (corresponding to $k=1$) in (18).□

Property 3. *In the space of the errors $e=y-f(\mathbf{x})$, the empirical risk obtained using the C-loss function behaves like the L_2 norm for small errors (samples correctly classified with high confidence). It behaves like the L_1 norm as the errors increase, and approaches the L_0 norm for very large errors (misclassified samples). The kernel size dictates the rate at which the empirical risk transitions from L_2 to L_0 behavior in the error space.*

Proof. The empirical risk using the C-loss function is

$$\hat{R}_C(f)=\beta\left(1-\frac{1}{n}\sum_{i=1}^{n}\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right) \tag{19}$$

$$\hat{R}_C(f)=\left(1-\exp\left(\frac{-1}{2\sigma^2}\right)\right)^{-1}\left(1-\frac{1}{n}\sum_{i=1}^{n}\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right) \tag{20}$$

Computing the limit as $\sigma\to\infty$, using L'Hospital rule,

$$\lim_{\sigma\to\infty}\hat{R}_C(f)=\lim_{\sigma\to\infty}\frac{\frac{1}{n}\sum_{i=1}^{n}\exp\left(\frac{-e_i^2}{2\sigma^2}\right)e_i^2}{\exp\left(\frac{-1}{2\sigma^2}\right)} \tag{21}$$

$$\lim_{\sigma\to\infty}\hat{R}_C(f)=\frac{1}{n}\sum_{i=1}^{n}e_i^2=\|e\|_2 \tag{22}$$

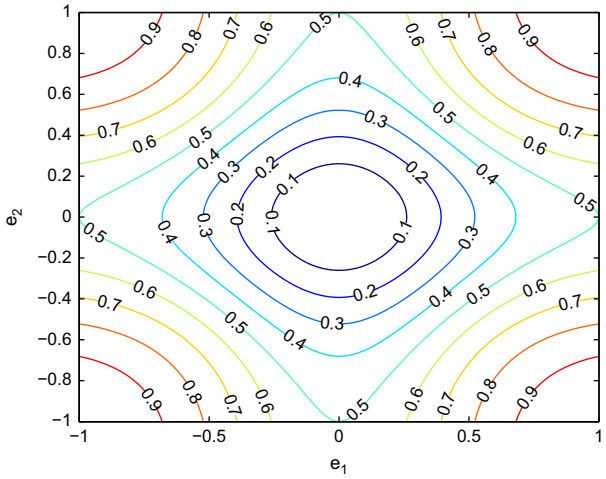

Fig. 3. The empirical risk function in a 2D space of errors $e=y-f(x)$, obtained using the C-loss function, with $\sigma=0.5$. Near the origin, the behavior of the risk is similar to the L_2 norm. Further away, it transitions to the L_1 norm. For large errors, the risk function approaches the L_0 or the counting norm.

Therefore, as σ increases, the empirical risk behaves like an L_2 norm in most of the space. Now, consider the limit as $\sigma\to 0^+$,

$$\lim_{\sigma\to 0^+}\hat{R}_C(f)=\lim_{\sigma\to 0^+}\frac{\left(1-\frac{1}{n}\sum_{i=1}^{n}\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right)}{\left(1-\exp\left(\frac{-1}{2\sigma^2}\right)\right)} \tag{23}$$

Using the fact that the limit of the denominator is 1, we obtain

$$\lim_{\sigma\to 0^+}\hat{R}_C(f)=\lim_{\sigma\to 0^+}\left(1-\frac{1}{n}\sum_{i=1}^{n}\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right) \tag{24}$$

$$\lim_{\sigma\to 0^+}\hat{R}_C(f)=\frac{1}{n}\sum_{i=1}^{n}\lim_{\sigma\to 0^+}\left(1-\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right) \tag{25}$$

Now,

$$\lim_{\sigma\to 0^+}\left(1-\exp\left(\frac{-e_i^2}{2\sigma^2}\right)\right)=\begin{cases}0 & \text{if } e_i=0\\ 1 & \text{if } e_i\neq 0\end{cases} \tag{26}$$

Therefore the limit in (25) is essentially a count of the number of non-zero error samples, or the L_0 norm of the errors.

$$\lim_{\sigma\to 0^+}\hat{R}_C(f)=\|e\|_0 \tag{27}$$

The kernel width (or, equivalently, the distance from the origin in the space of errors) governs the nature of the empirical risk obtained using the C-loss function. Fig. 3 depicts the transition from an L_2 norm like behavior close to the origin, to L_0 behavior far away, with an L_1 like region in between. □

Property 4. *For properly selected values of the kernel width, the C-loss function becomes less sensitive to the 'hard' or 'confusing' samples near the decision boundary, leading to decision boundaries that are robust to overfitting.*

Justification: Let us compute the sensitivity of the C-loss function with respect to the margin, or equivalently the error $e=y-f(\mathbf{x})$, by evaluating the following derivative:

$$\frac{\partial l_C(e)}{\partial e}=\frac{\partial}{\partial e}\beta\left[1-\exp\left(\frac{-e^2}{2\sigma^2}\right)\right] \tag{28}$$

$$\frac{\partial l_C(e)}{\partial e}=\frac{\beta}{\sigma^2}\exp\left(\frac{-e^2}{2\sigma^2}\right)e. \tag{29}$$

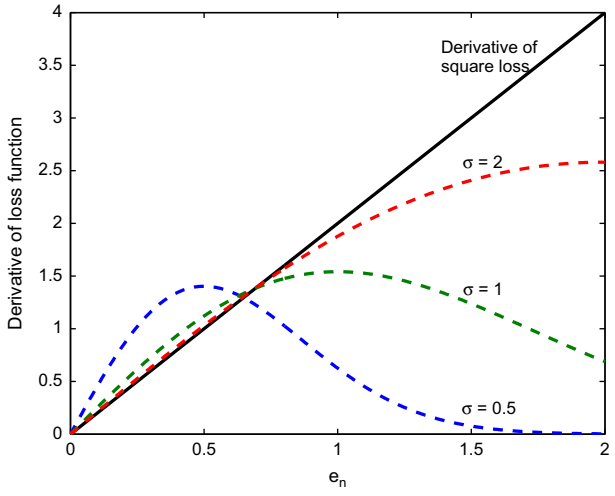

Fig. 4. The derivative of the C-loss function (dotted lines) for different values of kernel width σ. The derivative of the squared loss function (solid black plot) is also shown. Samples which produce higher errors influence the weight update more if the square loss function is used. However, in case of the C-loss function, highly erroneous samples have less influence on the weights (the decision boundary), depending on the kernel width σ.

Since $e = y - f(\mathbf{x})$, it is easy to see that if $1 < |e| < 2$, the sample is misclassified, and if $0 < |e| < 1$, the sample is correctly classified. $|e| = 1$ represents the decision boundary. Fig. 4 shows the plots of the sensitivity with respect to e, for different values of the kernel width σ. The derivative of the square loss is also shown for comparison. It can be seen that the kernel width σ controls the sensitivity of the C-loss function over the error dynamic range, i.e. its peak value and where it occurs in the space of the errors. The samples within the main peak have a high influence on learning the decision boundary. Thus, for $\sigma = 2$, the large errors control totally the placement of the discriminant function, with all issues that are well known (high sensitivity to outliers, and overtraining that occurs when the system has many free parameters and produces separation surfaces that have high curvature). When the sensitivity is centered at $|e| < 1$, essentially the centroid of the class controls the placement of the discriminant function, not the samples near the boundary. Moreover, the sensitivity is attenuated far away in the erroneous samples region, so the discriminant function is less likely to overfit these samples, even on prolonged training.

It is interesting to notice from Figs. 1 and 2 that the C-loss function with $\sigma = 1$ provides an approximation to the hinge loss function, and has the advantage that large errors are not weighted as high as in case of the square loss function. However, looking at Fig. 4, for $\sigma = 0.5$, even the samples near the decision boundary ($|e| = 1$) are attenuated and the learning process becomes less sensitive to these 'difficult' samples. This is beyond what can be achieved with the hinge loss (or any convex loss). The C-loss function aims at doing exactly this. Ignoring the 'confusers' near the class boundaries can prevent overfitting.

Contrary to this approach, classifiers like AdaBoost place exponentially large weightage to the difficult samples near the boundary. While such an approach has enjoyed success in classifying low noise and well separated classes, in the case of noisy and highly overlapping classes, the AdaBoost algorithm is known to overfit and yield poor generalization [26–28]. In fact, pruning such difficult samples in the training set has been successfully used as a way of regularizing the solution and improving the generalization performance (by preventing overfitting) of classifiers like AdaBoost [29,30]. However, instead of using additional criteria for ranking the samples based on their hardness and selecting thresholds for pruning the dataset (as done in [30]), the proposed loss function automatically achieves the same effect by simply choosing a suitable value of the kernel width parameter, based on Fig. 4. Our simulations in Section V corroborate this claim. We show that even after prolonged training using the C-loss function with $\sigma = 0.5$, the generalization performance of the classifier is well maintained. This is a significant advantage over traditional convex loss functions, which resort on heuristic techniques like early stopping of training to obtain better generalization.

Table 1
Specifications of datasets used in our evaluations. n=number of samples, f=number of features, c=number of classes, n_{tr}=number of samples used for training.

Dataset name	n	f	c	n_{tr}
Wisconsin Breast Cancer	683	9	2	200
Pima Indians Diabetes	768	8	2	300
Connectionist Bench	208	60	2	100
BUPA Liver Disorder	345	6	2	100
Cleveland Heart Disease	297	13	5	100
Blood Transfusion	748	5	2	300
Vowel Context	990	10	11	300
Image Segmentation	2310	19	7	500
Waveform	5000	21	3	1000

4. Training using C-loss function

The C-loss function (for $\sigma < 1$) is a non-convex function of the margin. Therefore it is difficult to obtain the optimal discriminant function f using convex optimization techniques. However, since the C-loss is always a smooth function, gradient based procedures can still be utilized.

Fig. 3 shows that unlike the square loss and most metric spaces (where the metric is constant), the C-loss function is local, i.e. the sensitivity to the structure of the space is 'local' to the operating point, where the locality is determined by the kernel width. Such localization of sensitivity means that a gradient based optimization procedure would be very slow to converge, if initialized far away from the optimal solution. One way of avoiding such slow convergence is to first approach the vicinity of the global optimum using a global, convex loss function like the square loss or the logistic loss, and then switch to the C-loss function [12]. This is equivalent to using a loss function of the form:

$$l(y, f(\mathbf{x}), m) = \mathbf{I}_{m < N_s} l_g(y, f(\mathbf{x})) + (1 - \mathbf{I}_{m < N_s})\beta[1 - \kappa_\sigma(y - f(\mathbf{x}))],$$

where m denotes the current epoch[2] of training, and $\mathbf{I}_{m < N_s}$ is an indicator function that is equal to one if m is less that a predefined constant N_s, and zero otherwise. $l_g(y, f(\mathbf{x}))$ stands for a global, convex loss function such as the square loss or the logistic loss.

In the above formulation, however, it is difficult to know the best value of N_s, that is the number of epochs after which the loss function should switch. Based on our experience, there is an optimum value of N_s that is data and architecture dependent, and can be determined by cross validation.

To alleviate this problem to some extent, instead of having a hard switching threshold like above, we propose to use a soft switch between the two loss functions. We therefore define the C-loss function to be a function of m as follows:

$$l_C(y, f(\mathbf{x}), m) = (1 - \gamma_m) l_g(y, f(\mathbf{x})) + \gamma_m \beta[1 - \kappa_\sigma(y - f(\mathbf{x}))], \tag{30}$$

where $\gamma_m = m/N$, and N is the total number of training epochs. This means that the weighting of the loss functions in the above equation linearly varies from 0 to 1 (and from 1 to 0) over the total number of training epochs. When $\gamma_m = m/N$, the high weighting on the convex loss function l_g initially helps avoid local optima and approach the vicinity of the global optimum. In the vicinity of the optimum solution (in the latter stages of training), the effect of the convex loss diminishes and the useful properties of the C-loss function (particularly Property 4) begin to hold. We show in our results that the classification performance using the C-loss function as described by (30) remains consistent across a wide range of values of N, unlike the traditional square loss or logistic loss function.

We use the C-loss function as described by (30) in all the experiments that follow.

5. Experiments and results

We carry out our experiments using datasets obtained from the UCI Machine Learning Repository [21]. The specifications of the datasets used are tabulated in Table 1.

We use the C-loss function for training single hidden layer perceptrons and RBF networks using backpropagation.

Our evaluations are divided into two parts. In the first part (Section 5.1), we analyze in detail the classification performance of the C-loss function when system parameters such as number of processing elements (PEs) and number of training epochs are

[2] An epoch here denotes one pass of the entire training set. Therefore, if there are N_t samples in the training data, each epoch consists of N_t sample updates.

varied in the network. Such an analysis is important since the choice of these parameters is critical in practical applications, and can often lead to problems like overfitting. We show that the use of the C-loss function reduces the variability of classification performance across these parameters, and therefore it should be the preferred loss function while training networks for classification problems.

In the second part (Section 5.2), we use cross validation to obtain the optimum set of network parameters and compare the optimum performance yielded by the C-loss function, to those obtained by other classifiers. Although the primary objective of proposing the C-loss function is to improve the performance of neural network classifiers, in order to see how the C-loss trained networks compare to the modern state-of-the-art methods we also compare our results to the soft margin SVM (using Gaussian kernels).

5.1. Performance across system parameters

We motivate our analysis by first qualitatively evaluating the C-loss function on some simple synthetic datasets. Fig. 5 shows a comparison of the discriminant functions obtained by the C-loss function and the square loss function, after training a single hidden layer perceptron with 20 processing elements (PEs) in the hidden layer. In this example, both the classes have Gaussian distributions with the same covariance structure. The datasets are normalized to have unit variance along each feature. In such a case, the Bayes' optimal discriminant is a straight line, perpendicularly bisecting the line joining the centers of the Gaussians. The C-loss function leads to a closer approximation of the optimal discriminant, whereas the square loss function tends to overfit the training samples, leading to poor generalization.

Fig. 6 shows another similar example. One of the classes is drawn from a Gaussian distribution, whereas the second class has a 'half moon' shape. Again, the square loss function creates a discriminant function that is overfitted and too specific to the training samples. The C-loss function produces a discriminant that does not tend to overfit.

Motivated by these preliminary observations, we now quantitatively explore in more detail the robustness of classification performance to overfitting, when various system parameters like number of PEs, number of training epochs are varied in the network. We consider three real world datasets for this analysis – *Pima Indians Diabetes*, *Wisconsin Breast Cancer* and *Connectionist Bench*.

Fig. 7 shows plots of the classification (generalization) performance of the C-loss functions with $\sigma = 0.5$ and $\sigma = 1$, and the square loss function on the *Diabetes* dataset. Each plot shows the classification performance at different training epochs. The total number of training epochs N is varied along the rows and the number of hidden layer processing elements (PEs) is varied along the columns. Furthermore, each subplot is obtained by averaging over 100 Monte Carlo runs, with random initial weights and random sampling of the training data. Fig. 7 therefore shows the performance of the C-loss and square loss functions across 9 different combinations of the two system parameters: the total number of training epochs N and number of hidden layer PEs.

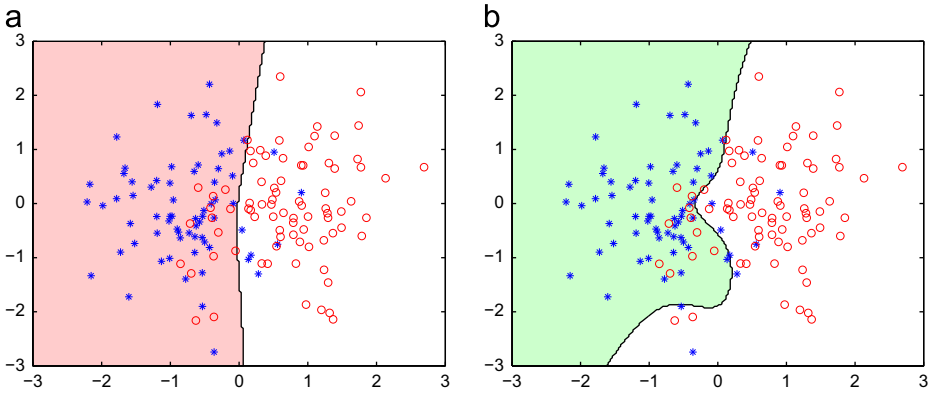

Fig. 5. Discriminant functions obtained while classifying a synthetic dataset. Both classes have a Gaussian distribution with the same covariance structure. (a) Discriminant function obtained using the C-loss function and (b) discriminant function obtained using the conventional square loss function. The square loss function tends to cause overfitting.

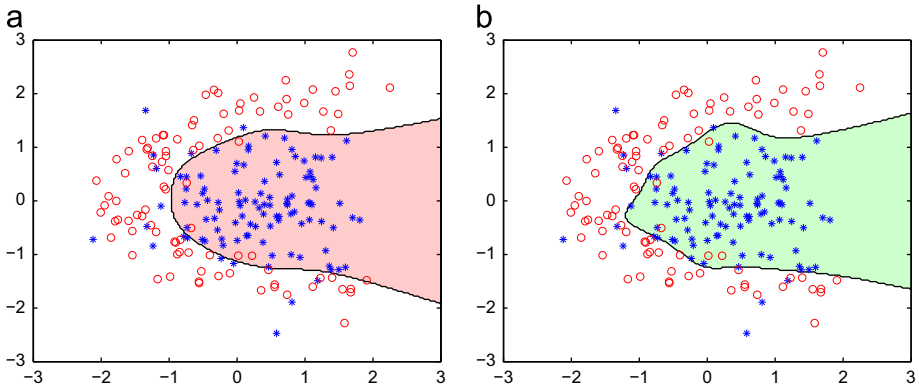

Fig. 6. Discriminant functions obtained while classifying a synthetic dataset. One of the classes is drawn from a Gaussian density and the other has a 'half moon' structure. (a) Discriminant function obtained using the C-loss function and (b) discriminant function obtained using the conventional square loss function. The square loss function causes overfitting.

Figs. 8 and 9 show similar results for the *Breast Cancer* and *Connectionist Bench* datasets, respectively. Note that for the *Connectionist Bench* dataset, we first reduced its dimensionality from 60 to 10, since all the classifiers we tested performed better in the reduced space. Although several methods for dimensionality reduction exist in the literature [31–34], we used the Chernoff criterion based method proposed in [32] for this preprocessing.

A clear trend can be observed from the plots for all three datasets: the generalization performance of the square loss function attains a peak, and then drops if the network is trained further. However, the C-loss function does not tend to overfit the training data even after prolonged training, as discussed in Property 4. As the number of PEs is increased, the square loss function becomes more prone to overfitting. However, the C-loss function is more consistent across network topologies.

Table 2 tabulates the generalization performance obtained *at the end of training*, for each of the 9 cases of Fig. 7, for the *Diabetes* dataset. Among these, the best classification performance for the C-loss functions ($\sigma = 0.5$ and $\sigma = 1$) and the square loss function are shown in bold. The plots and the tabulated results also reconfirm that C-loss with $\sigma = 0.5$ is a better kernel size. Tables 3 and 4 show similar tabulations from Figs. 8 and 9, for the *Breast Cancer* and *Connectionist Bench* datasets, respectively.

5.1.1. Comparing variance in generalization performance

It is clear from Figs. 7–9 that the C-loss function yields higher generalization performance than the square loss function, across all the 9 different combinations of the two system parameters – number of training epochs and number of hidden layer PEs in the network. In Table 5, we compute the variance of the generalization performance over these 9 configurations of system parameters. That is, we compute the variance along the rows of Tables 2–4. We observe that the C-loss function (particularly with $\sigma = 0.5$) not only produces better generalization than the square loss, but it is also more consistent (having less variance) across changes in system parameters, as shown in Table 5 . In other words, the C-loss function is less sensitive to the choice of training parameters (number of training epochs and hidden layer PEs) as compared to the square loss. The C-loss function therefore helps in decoupling to some extent the network's generalization performance from its training parameters.

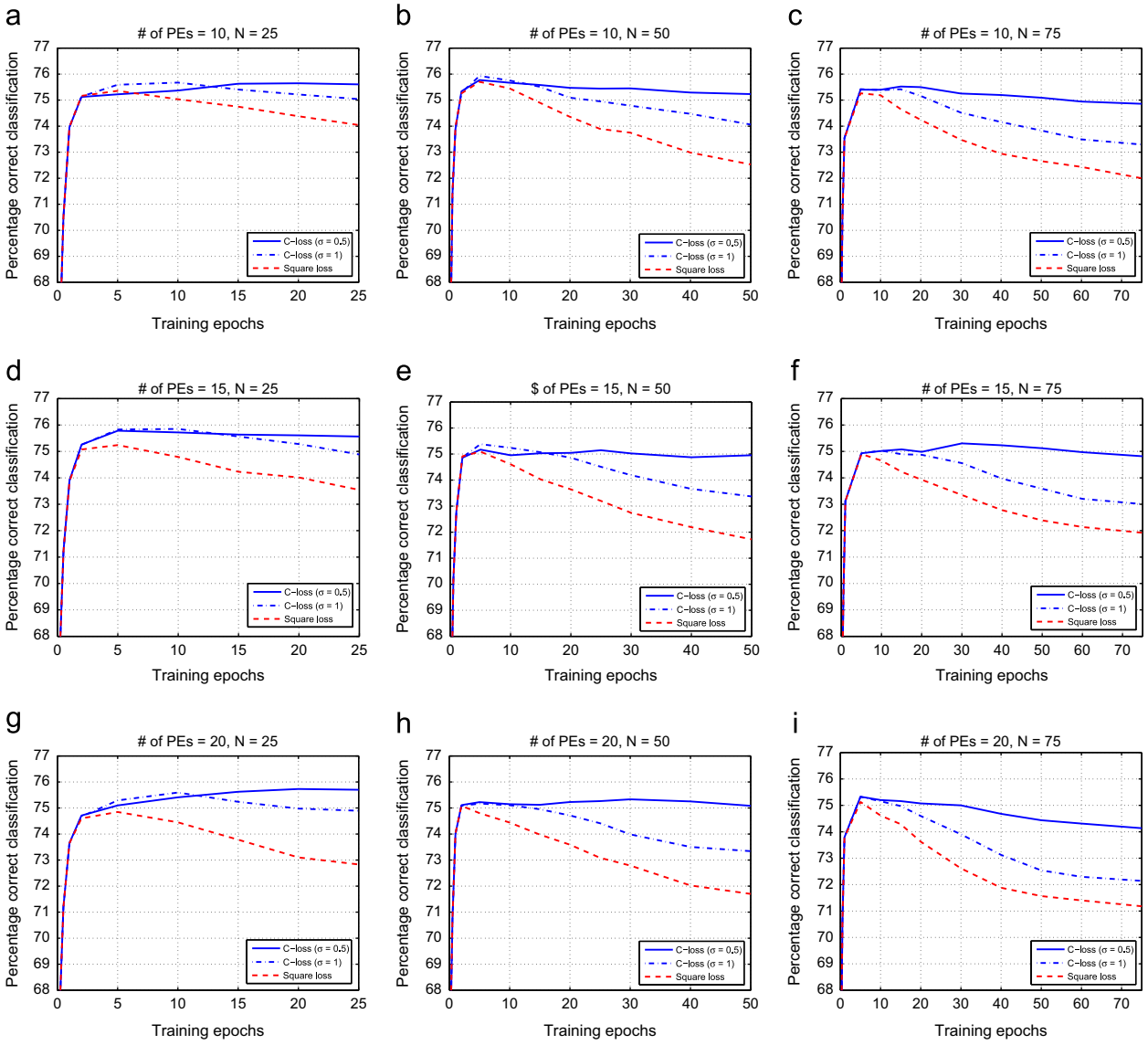

Fig. 7. Plots of the mean classification performance of the C-loss and the square loss functions on the Diabetes dataset, for different numbers of PEs (10, 15 and 20) and different total training epochs (N=25, 50, 75). The number of PEs is varied along the columns, and the number of total training epochs (N) is varied along the rows. The square loss function tends to overfit the data with more training, resulting in a drop in generalization performance. The C-loss function remains relatively consistent over the number of training epochs and the number of PEs in the network.

5.1.2. Best results over Monte Carlo trials

The plots in Figs. 7–9 and the values in Tables 2–4 have shown the *average* generalization performance over 100 Monte Carlo runs with random initialization of network weights and random sampling of the training data. The averaged result reflects the likelihood outcome of a naive user using the proposed method. However, since we are proposing a new loss function, we would also like to determine its *best* possible performance attainable across the Monte Carlo trials. We do this as follows: We first perform 100 Monte Carlo runs in which the neural network is trained using randomly chosen training samples and random initial weights. Out of these, we select the configuration that gives the *best* classification performance after testing on test dataset. This entire process is performed 100 times and an average of these best results is presented.

Fig. 10 shows the plots of best generalization performance on the *Diabetes* dataset, *Breast Cancer* dataset and *Connectionist Bench* dataset. The best results are attained when the neural network uses 20 hidden layer PEs and $N=20$ for *Diabetes* dataset, 10 hidden layer PEs and $N=50$ for *Breast Cancer* dataset and, 20 hidden layer PEs and $N=75$ for *Connectionist Bench* dataset. The final results obtained at the end of training are presented in Table 6.

5.1.3. Training RBF networks with C-loss

We now train RBF networks using the C-loss function, and compare it to the logistic loss function. We randomly choose 50 samples from the training set to be RBF centers. We use isometric Gaussians as the activation functions. At the output layer, we use the C-loss function of (30) to train, but now, l_g is the logistic loss function defined as

$$l_g(y,f(\mathbf{x})) = \log(1+e^{-yf(\mathbf{x})}). \tag{31}$$

We use the C-loss function with l_g defined as above and compare it to using the logistic loss function alone. Once the RBF centers are chosen, the 'hidden' layer of the RBF network remains fixed while training, and the output layer weights are adjusted using simple gradient descent. The learning algorithm is therefore simpler than a full blown backpropagation algorithm.

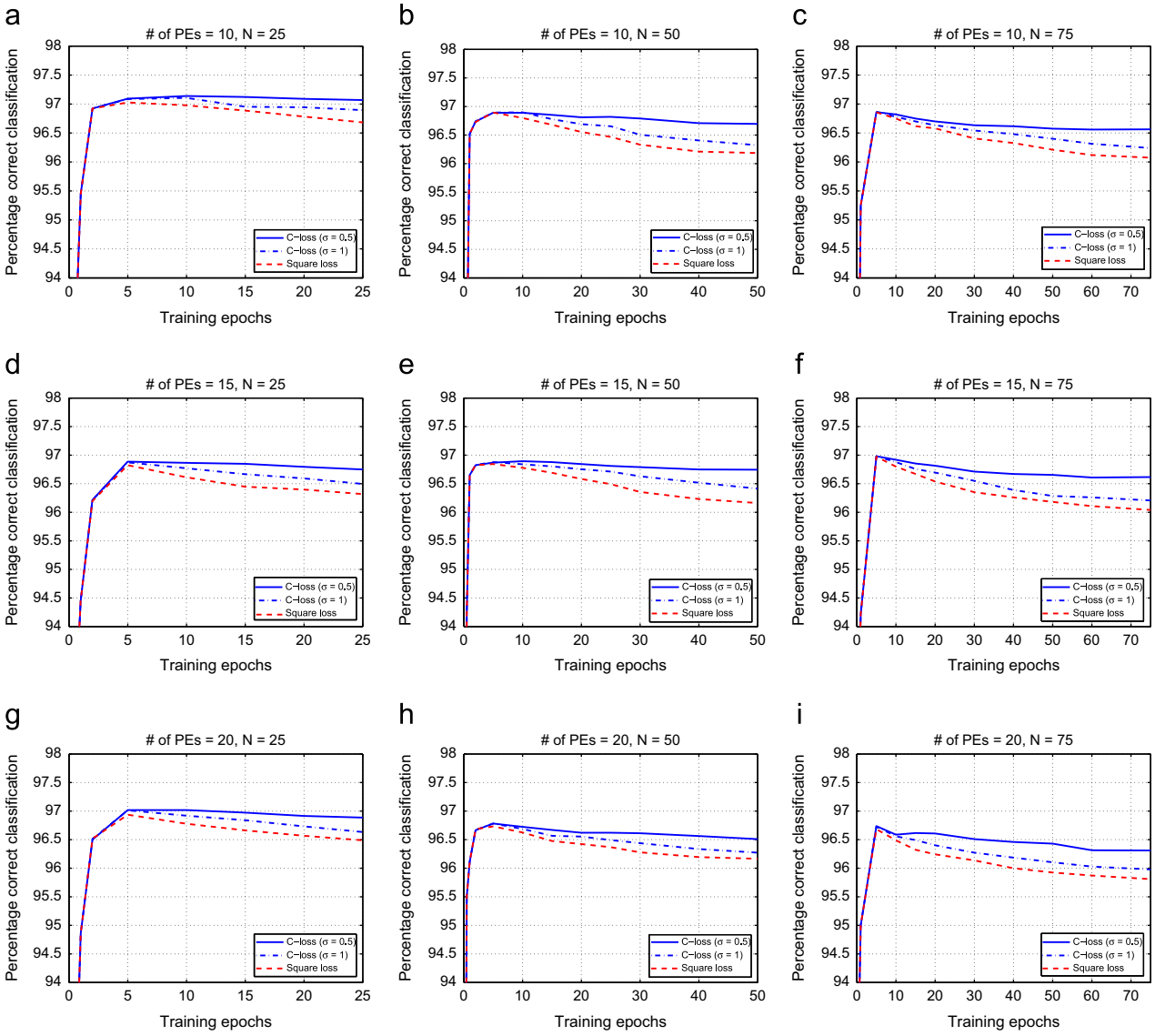

Fig. 8. Plots of the mean classification performance of the C-loss and the square loss functions on the Breast Cancer dataset, for different numbers of PEs (10, 15 and 20) and different total training epochs ($N=25$, 50, 75). The number of PEs is varied along the columns, and the total training epochs N is varied along the rows. The square loss function tends to overfit the data with more training, resulting in a drop in generalization performance. The C-loss function remains relatively consistent over the number of training epochs and the number of PEs in the network.

As before, we compare the generalization performance (on the test set) during several stages of training. Our results on the *Diabetes*, *Breast Cancer*, and *Connectionist Bench* datasets are shown in Fig. 11. We have used the same training/testing dataset division as in the experiments with MLPs. To avoid being sensitive to a particular selection of the training set, and to avoid sensitivity to a particular choice of RBF centers, we have repeated the training over 100 Monte Carlo trials, with random training data and randomly selected RBF centers. Fig. 11 shows the average results across these 100 trials.

We observe a similar trend as in the MLP case. Although the logistic loss is perhaps slightly more robust than the square loss,

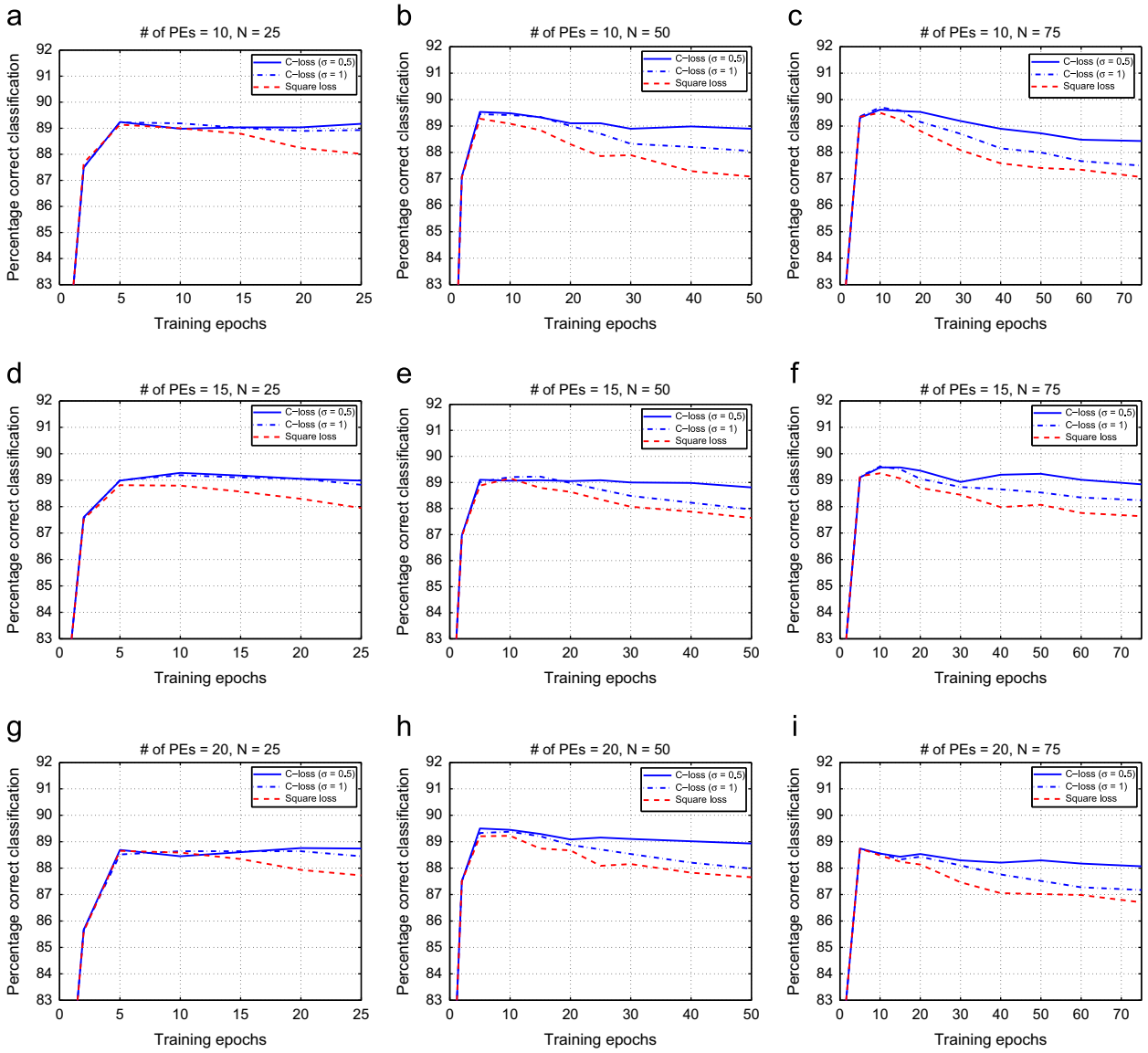

Fig. 9. Plots of the mean classification performance of the C-loss and the square loss functions on the Connectionist Bench data, for different numbers of PEs (10, 15 and 20) and different total training epochs (N=25, 50, 75). The number of PEs is varied along the columns, and the total training epochs N is varied along the rows. The square loss function tends to overfit the data with more training, resulting in a drop in generalization performance. The C-loss function remains relatively consistent over the number of training epochs and the number of PEs in the network.

Table 2
Generalization performance (in percent) on the DIABETES dataset, at the end of training using the C-loss and the square loss functions, for different numbers of PEs and training epochs.

Number of PEs	10			15			20		
Total training epochs, N	25	50	75	25	50	75	25	50	75
C-loss ($\sigma = 0.5$) performance	75.66	75.23	74.86	75.56	74.95	74.82	***75.70***	75.08	74.14
C-loss ($\sigma = 1$) performance	***75.04***	74.06	73.29	74.88	73.37	73.00	74.89	73.34	72.14
Square-loss performance	***74.04***	72.53	72.00	73.54	71.72	71.92	72.83	71.69	71.19

Table 3
Generalization performance (in percent) on the breast cancer dataset, at the end of training using the C-loss and the square loss functions, for different numbers of PEs and total training epochs.

Number of PEs	10			15			20		
Total training epochs, N	25	50	75	25	50	75	25	50	75
C-loss ($\sigma = 0.5$) performance	***97.07***	96.70	96.56	96.75	96.75	96.62	96.89	96.51	96.31
C-loss ($\sigma = 1$) performance	***96.89***	96.32	96.24	96.5	96.41	96.21	96.63	96.27	96.58
Square-loss performance	***96.68***	96.19	96.07	96.32	96.16	96.04	96.49	96.17	95.81

Table 4
Generalization performance (in percent) on the connectionist bench dataset, at the end training of using the C-loss and the square loss functions, for different numbers of PEs and total training epochs.

Number of PEs	10			15			20		
Total training epochs, N	25	50	75	25	50	75	25	50	75
C-loss ($\sigma = 0.5$) performance	***89.17***	88.9	88.43	88.98	88.81	88.94	88.74	88.93	88.07
C-loss ($\sigma = 1$) performance	***88.93***	88.05	87.50	88.83	87.97	88.24	88.45	87.78	87.17
Square-loss performance	***88.02***	87.09	87.07	87.95	87.64	87.64	87.72	87.66	86.71

Table 5
Variance of generalization performance of the C-loss function ($\sigma = 0.5$ and $\sigma = 1$) and square loss function, across the 9 different combinations of training parameters.

Dataset	Diabetes	Breast Cancer	Conn. Bench
C-loss ($\sigma = 0.5$) variance	***0.2477***	***0.0488***	***0.1227***
C-loss ($\sigma = 1$) variance	1.0010	0.0498	0.1807
Square-loss variance	0.8762	0.0655	0.1936

the generalization performance nonetheless tends to suffer on excessive training. The use of the C-loss function improves the generalization. Note that the performance of the C-loss in this experiment was found to be roughly the same for both $\sigma = 1$ and $\sigma = 0.5$.

We do not show the performance variation across system parameters like number of RBF centers, but we have observed similar trends.

5.2. Further evaluation

In the previous results we have analyzed the performance variation of the C-loss function across network parameters like number of PEs and number of training epochs using three datasets, and found that the C-loss yields results that are much more robust across these parameters. In this section, we use more datasets and compare the best performance of the C-loss function after choosing the optimal network parameters using cross validation.

While training MLPs, we use cross validation to select the optimum number of hidden layer PEs and the number of training epochs for each dataset. Similarly for training RBF networks, we use cross validation to choose the optimum number of basis functions, and the number of training epochs. We also compare our results to those obtained by soft margin (Gaussian) kernel SVMs. SVMs require setting a box constraint parameter, C, which is effectively a regularization parameter. We use cross validation to choose the best values of C and also the Gaussian kernel bandwidth parameter, for each dataset.

Table 7 summarizes our results. We have used $\sigma = 0.5$ in the C-loss function for the results in this section. We observe that in all nine datasets, the proposed C-loss function consistently and significantly improves over the square loss and the logistic loss when used to train MLPs and RBF networks, respectively. This clearly demonstrates the robustness achieved by the C-loss over the other losses while training networks. We also see that the performance achieved by using the C-loss function is comparable to those of SVM, and in some datasets (such as *Pima Indians Diabetes*, *Cleveland Heart Disease* and *Blood Transfusion*), the C-loss has a notably better performance. The best result for each dataset is shown in bold.

6. Discussion

6.1. Applicability

We have seen in this paper that using the C-loss function is robust to prolonged training, particularly on noisy and difficult datasets like *Diabetes*. Overfitting is a serious problem since the number of training epochs or the capacity of the model is never 'optimally' known, and it is always easy and tempting to overtrain a model, with the hope of better results. If the class of discriminant functions is highly restricted, or if the training is done for a very small number of epochs, *underfitting* would be the cause of suboptimal performance. In the case of underfitting, the C-loss function is not expected to improve over the base loss function. The C-loss function is best applicable when overfitting is expected to occur, while training unconstrained models on noisy datasets, over a large number of epochs.

6.2. Choice of kernel size

We have shown that the sensitivity of the C-loss function in the error space is high only within a window, controlled by the kernel width. Therefore, since the parameters of the discriminant function are not sensitive to samples outside this window, it prevents overfitting. A question naturally arises: what should be the best value of this kernel width? It is important to note that in any task that uses kernels to weight data, the kernel size should always be chosen relative to the dynamic range of the variable or signal on which the kernel operates. In case of the C-loss function, this

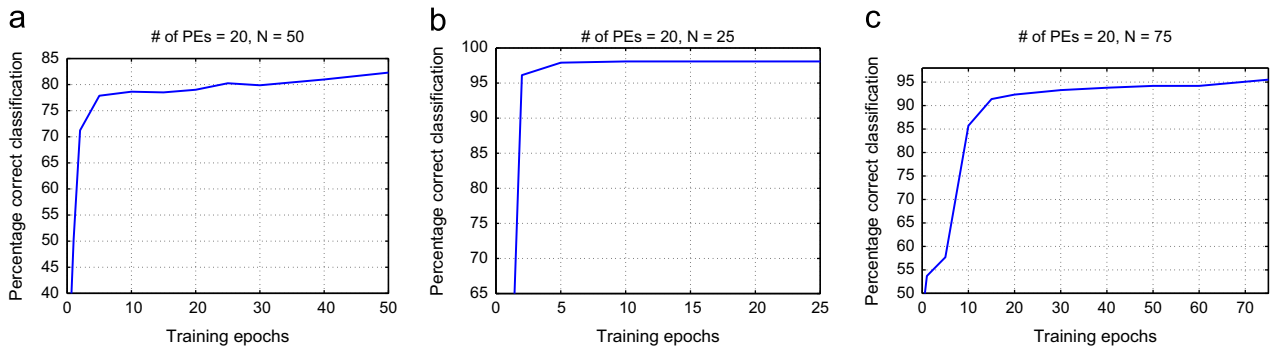

Fig. 10. Evolution of generalization performance of the C-loss function over training epochs, when the *best* performance is picked out of 100 Monte Carlo trials with random initial weights and random sampling of training data. (a) Diabetes dataset, (b) Breast Cancer dataset, and (c) Connectionist Bench dataset.

Table 6
Best results obtained from 100 Monte Carlo runs with random initial weights and random sampling of training data, using the C-loss function, for the three datasets.

Dataset	Diabetes PEs=20, N=50	Breast Cancer PEs=20, N=25	Conn. Bench PEs=20, N=75
Best performance out of 100 Monte Carlo trials	82.29	98.07	95.53

variable is the classification margin or error. Clearly, a very high value of the kernel width with respect to this error would essentially reduce the C-loss function to the square loss, and it would lose its benefits. In a classification task, the range of the absolute value of the errors is fixed, and known *a priori* (either 0 to 2 or 0 to 1). This makes the task of choosing the kernel size in classification much easier as compared to regression tasks such as in [25,35]. Within this restricted range as well, more intuition on choosing the kernel size is obtained by looking at Fig. 4, which shows how the kernel width controls sensitivity of the C-loss function in the space of errors. Choosing $\sigma = 0.5$ places most of the weight within the 0–1 range of the error, which corresponds to the correctly classified samples. Choosing this kernel size should provide the most robustness since outlying samples (high error) as well as confusing samples (error around 1) are downweighted. We have experimentally shown that as compared to a bigger kernel like $\sigma = 1$, choosing $\sigma = 0.5$ does indeed yields better results, particularly while training MLPs. We therefore recommend using the value $\sigma = 0.5$ to any user of the proposed method.

6.3. Choice of switching scheme

As an alternative to having a hard switching scheme as done before in [12], in this paper we proposed a cost function that smoothly (linearly) varies from a convex loss function to the C-loss function over the training epochs, in order to avoid local optima. This means that the longer the classifier is trained, the longer is the training period in which the convex loss dominates, and the closer the solution gets to the optimal as dictated by the convex loss. However, as the weighting on the C-loss function increases, there is always an improvement in generalization over the convex loss, irrespective of how long the classifier is trained. This improvement may be small in case of clean and well separated datasets, or may be larger in case of highly noisy datasets like the Diabetes dataset.

If using a hard switching scheme, in addition to choosing the number of training epochs, one also has to explicitly choose when to switch the loss function. The use of the proposed 'soft' switching scheme avoids making this explicit choice. Although it is difficult to theoretically compare the two schemes, we propose the 'soft' switching scheme as another way of incorporating the C-loss function in training. Both are valid ways of improving over the performance of conventional convex losses. Alternatively, the kernel size can also be 'annealed' during training, to take advantage of the invex properties of the C-loss. We plan to examine this and the other switching alternatives in greater detail in a follow up publication.

7. Conclusion

We have presented a loss function for classification that is inspired by a robust similarity measure, Correntropy. Correntropy induces a loss function that is smooth and non-convex and effectively approximates the L_0 loss for samples with high errors, and the L_2 loss for small errors.

We have tested the proposed loss function on some noisy, real world datasets. Instead of simply showing the 'best classification results' of the proposed loss function on a larger number of datasets, we chose to study deeper the behavior of the loss function as the system parameters are varied. It is more interesting and informative to note how consistent is the classification performance of the C-loss function (particularly in comparison to the square loss) across parameters like number of PEs and the total number of training epochs, than comparing the absolute best results of these classifiers in various datasets. Many classification algorithms often underperform due to improper choice of such parameters, or require additional techniques like cross validation to scan for the best set of parameters. Using the proposed loss function, this problem is alleviated.

Our simulations have been restricted to using simple first order gradient descent method for training the networks. However, the C-loss function can be used with online second order methods like the Levenberg Marquadt algorithm as well, and we expect the same behavior.

An important appeal of the C-loss function is that its computational complexity is the same as that of using the traditional square loss or logistic loss. Therefore, the proposed algorithm is a simple and practical way of improving any conventional neural network based classifier, without resorting to more computationally intensive cost functions like the minimization of the error entropy [36]. A potential and important class of applications is for big data, where the online nature of gradient descent can be used by stochastically sampling the data, i.e. without requiring the availability of all the data in memory, nor the use of huge matrices as in SVMs.

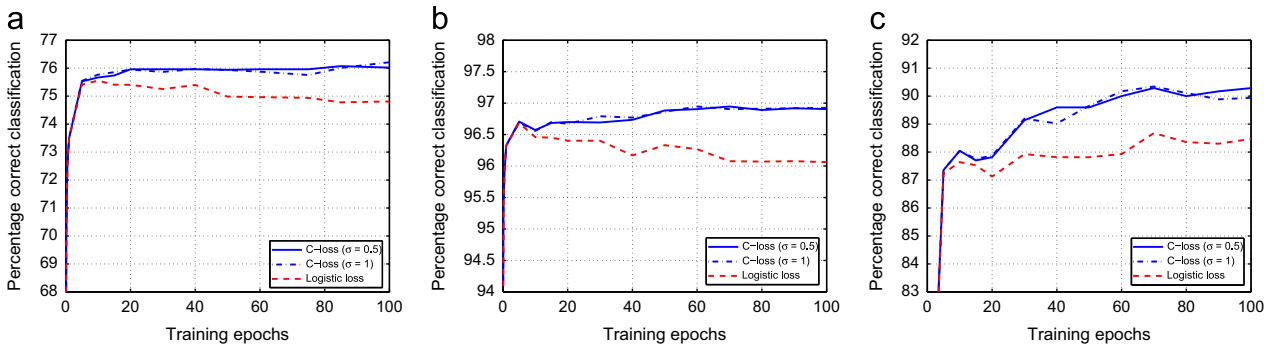

Fig. 11. Mean classification performance of the C-loss function ($\sigma=0.5$, and $\sigma=1$), and the logistic loss function, while training an RBF network. The plots show generalization performance at different training epochs. Each curve is obtained after averaging over 100 Monte Carlo trials, with randomly chosen training set, RBF centers, and initial weights. (a) Diabetes dataset, (b) Breast Cancer dataset, and (c) Connectionist Bench dataset.

Table 7
Classification results obtained using C-loss (on MLP and RBF network), square loss (on MLP), logistic loss (on RBF network) and SVM. The best result for each dataset is shown in bold.

Dataset name	MLP		RBF network		SVM
	Square loss	C-loss	Logistic loss	C-loss	
Wisconsin Breast Cancer	96.68	***97.07***	96.06	96.90	96.92
Pima Indians Diabetes	74.04	75.70	74.81	***76.02***	74.46
Connectionist Bench	88.02	89.17	88.16	***90.29***	90.22
BUPA Liver Disorder	66.52	69.93	65.88	69.12	***70.12***
Cleveland Heart Disease	60.01	62.32	60.32	***62.84***	61.08
Blood Transfusion	77.40	***79.82***	75.96	77.93	78.02
Vowel Context	76.26	80.14	77.18	80.69	***80.98***
Image Segmentation	95.03	96.30	94.88	***96.38***	95.08
Waveform	83.80	***86.16***	82.97	85.70	85.89

Conflict of interest

None declared.

Acknowledgements

Abhishek Singh was supported by the Joan and Lalit Bahl Fellowship and the Computational Science and Engineering Fellowship at the University of Illinois at Urbana-Champaign. Jose Principe was supported by NSF grant ECCS 0856441.

Appendix A. Backpropagation using the C-loss function

Before deriving the backpropagation equations, the notation used to denote the variables in neural network is summarized below for convenience.

w_{jk}^n: The weight between the PE (processing element) k and the PE j of the *previous* layer, at the nth sample update.[3]

y_j^n: Output of the PE j of the *previous* layer, at the nth sample update.

$net_k^n=\sum_j w_{jk}^n y_j^n$: Weighted sum of all outputs y_j^n of the previous layer, at the nth sample update.

$g()$: Sigmoidal squashing function in each PE.

$y_k^n=g(net_k^n)$: Output of the PE k of the current layer, at the nth sample update.

$y^n\in\{\pm 1\}$: The true label (or desired signal), for the nth sample.

The weights of a multilayer perceptron (MLP) are updated by moving opposite to the gradient of the empirical risk computed using the C-loss function.

$$w_{jk}^{n+1}=w_{jk}^n-\mu\frac{\partial R_C(e,m)}{\partial w_{jk}^n}, \tag{A.1}$$

where the risk $R_C(e,m)$ is simply the sample average of the C-loss function of (30), computed over the training set, at the mth epoch of training. Using the chain rule, the above equation can be written as

$$w_{jk}^{n+1}=w_{jk}^n+\mu\frac{\partial R_C(e,m)}{\partial e_n}g'(net_k^n)y_j^n. \tag{A.2}$$

Since this is an online procedure, the derivative of the risk with respect to the error at nth sample update, e_n, is essentially the derivative of the C-loss function, evaluated at e_n. Therefore,

$$w_{jk}^{n+1}=w_{jk}^n+\mu\frac{\partial l_C(e,m)}{\partial e_n}g'(net_k^n)y_j^n, \tag{A.3}$$

where $l_C(e,m)$ is defined by (30). The above equation is the general rule for updating all the weights of the MLP, and it is called the Delta rule, written simply as

$$w_{jk}^{n+1}=w_{jk}^n+\mu\delta_k^n y_j^n, \tag{A.4}$$

where

$$\delta_k^n=\frac{\partial l_C(e,m)}{\partial e_n}g'(net_k^n) \tag{A.5}$$

Depending on the type of weights (belonging to output layer or hidden layer), the computation of δ_k^n above differs. For the output layer weights, the computation is as follows:

$$\delta_o^n=\frac{\partial l_C(e,m)}{\partial e_n}g'(net_o^n). \tag{A.6}$$

For the previous (hidden) layer, the 'deltas' are computed as

$$\delta_h^n=g'(net_h^n)\sum_{o=1}^{N_o}\delta_o^n w_{ho}^n, \tag{A.7}$$

where N_o is the number of output layer PEs. A two class classification problem can therefore be generalized to N_o number of classes, by having the true or 'desired' labels at each output PE according to a 'one class versus the rest' strategy.

(A.(6) and A.7) can be used to update or train the weights of a neural network classifier using the C-loss function.

[3] If the network is trained for a total of N epochs, and if there are N_t number of samples in the training set, then there are a total of $n=N_t\times N$ sample updates during the entire training procedure.

References

[1] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[2] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction, Springer, 2006.
[3] V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[4] C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (1998) 121–167.
[5] S.-H. Yang, B.-G. Hu, A stagewise least square loss function for classification, in: SIAM International Conference on Data Mining, 2008.
[6] Y. LeCun, Who is afraid of non-convex loss functions?, in: NIPS Workshop on Efficient Machine Learning, Vancouver, Canada (Videolecture), 2003.
[7] Y. Liu, X. Shen, Multicategory ψ-learning, Journal of American Statistical Association 101 (474) (2006) 500–509.
[8] R. Collobert, F. Sinz, J. Weston, L. Bottou, Trading convexity for scalability, in: 23rd International Conference on Machine Learning (ICML), 2006.
[9] Y. Bengio, Learning Deep Architectures for AI, Technical Report 1312, University of Montreal, Department of Computer Science and Operations Research, 2003.
[10] Y. Bengio, Y. LeCun, Scaling learning algorithms towards AI, in: L. Bottou, O. Chapelle, D. DeCoste, J. Weston (Eds.), Large Scale Kernel Machines, MIT Press, 2007.
[11] I. Santamaria, P. Pokharel, J. Principe, Generalized correlation function: definition, properties, and application to blind equalization, IEEE Transactions on Signal Processing 54 (06) (2006) 2187–2198.
[12] A. Singh, J. Principe, A loss function for classification based on a robust similarity metric, in: IEEE World Congress on Computational Intelligence (WCCI), International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 2010.
[13] W. Liu, P. Pokharel, J. Principe, Correntropy: properties and applications in non-Gaussian signal processing, IEEE Transactions on Signal Processing 55 (11) (2007) 5286–5298.
[14] A. Singh, J. Principe, Using correntropy as a cost function in linear adaptive filters, in: International Joint Conference on Neural Networks, Atlanta, USA, 2009.
[15] A. Singh, J. Principe, A closed form recursive solution for maximum correntropy training, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2010.
[16] D. Erdogmus, S. Han, A. Singh, Algorithms for entropy and correntropy adaptation with applications to linear systems, Information Theoretic Learning (2010) 141–179.
[17] A. Singh, Cost Functions for Supervised Learning Based on a Robust Similarity Metric, Master's Thesis, University of Florida, 2010.
[18] J. Xu, J. Principe, A pitch detector based on a generalized correlation function, IEEE Transactions on Audio, Speech and Language Processing 16 (8) (2008) 1420–1432.
[19] Y. Xinnan, H. Ren, Peak-gram: pitch detection based in generalized correlation function across multiple channels, in: The 7th IASTED International Conference on Signal Processing, Pattern Recognition and Applications, 2010.
[20] K.-H. Jeong, W. Liu, S. Han, E. Hasanbelliu, J. Principe, The correntropy MACE filter, Pattern Recognition 42 (2009) 871–885.
[21] K. Bache, M. Lichman, UCI machine learning repository, URL: ⟨http://archive.ics.uci.edu/ml⟩, 2013.
[22] P. Bartlett, M. Jordan, J. McAuliffe, Convexity, classification and risk bounds, Journal of American Statistical Association 101 (473) (2006) 138–156.
[23] Y. Lin, A Note on Margin-Based Loss Functions in Classification, Technical Report 1044, University of Wisconsin-Madison, Department of Statistics, 2001.
[24] E. Parzen, On the estimation of a probability density function and mode, Annals of Mathematical Statistics 33 (1962) 1065–1076.
[25] A. Singh, J.C. Príncipe, Information theoretic learning with adaptive kernels, Signal Processing 91 (2011) 203–213.
[26] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Machine Learning 40 (2000) 139–157.
[27] G. Rätsch, T. Onoda, K.-R. Müller, Soft margins for adaboost, Machine Learning 42 (2001) 287–320.
[28] Y. Sun, S. Todorovic, J. Li, Reducing the overfitting of adaboost by controlling its data distribution skewness, International Journal of Pattern Recognition and Artificial Intelligence 20 (2005) 1093–1116.
[29] A. Angelova, Y. Abu-Mostafa, P. Perona, Pruning training sets for learning of object categories, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 494–501.
[30] S. Merler, B. Caprile, C. Furlanello, I. Techrep, Bias-variance control via hard points shaving, International Journal of Pattern Recognition and Artificial Intelligence 18 (2002) 891–903.
[31] R. Duda, P. Hart, D. Stork, Pattern Classification, Wiley, 2001.
[32] R.P.W. Duin, M. Loog, Linear dimensionality reduction via a heteroscedastic extension of LDA: the chernoff criterion, IEEE TPAMI 26 (6) (2004) 732–739.
[33] M.S. Mahanta, A.S. Aghaei, K.N. Plataniotis, S. Pasupathy, Heteroscedastic linear feature extraction based on sufficiency conditions, Pattern Recognition 45 (2) (2012) 821–830.
[34] L. Rueda, M. Herrera, Linear dimensionality reduction by maximizing the chernoff distance in the transformed space, Pattern Recognition 41 (10) (2008) 3138–3152.
[35] A. Singh, J. Principe, Kernel width adaptation in information theoretic cost functions, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2010.
[36] J. Santos, L. Alexandre, J. Marques de Sa, The error entropy minimization algorithm for neural network classification, in: International Conference on Recent Advances in Soft Computing, 2004.

Abhishek Singh is a currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He received an M.S. degree in Electrical and Computer Engineering from the University of Florida, Gainesville in 2010, and the B.Tech degree from Dhirubhai Ambani Institute of Information and Communication Technology, India in 2008. He is the recipient of the Joan and Lalit Bahl Fellowship and the Computational Science and Engineering Fellowship for interdisciplinary and computation-oriented research at the University of Illinois. His research interests include statistical signal processing, pattern recognition and machine learning and application in computer vision and image analysis problems.

Rosha Pokharel is currently a Ph.D. student in the Computational NeuroEngineering Laboratory and the Department of Electrical and Computer Engineering at the University of Florida, Gainesville.

Jose C. Principe is a Distinguished Professor of Electrical and Biomedical Engineering at the University of Florida since 2002. He joined the University of Florida in 1987, after an eight year appointment as a Professor at the University of Aveiro, in Portugal. Dr. Principe holds degrees in electrical engineering from the University of Porto (Bachelor), Portugal, University of Florida (Master and Ph.D.), USA and a Laurea Honoris Causa degree from the Universita Mediterranea in Reggio Calabria, Italy. Dr. Principe interests lie in non-linear non-Gaussian optimal signal processing and modeling and in biomedical engineering. He created in 1991 the Computational NeuroEngineering Laboratory to synergistically focus the research in biological information processing models. He recently received the Gabor Award from the International Neural Network Society for his contributions.

Dr. Principe is a Fellow of the IEEE, Fellow of the AIMBE, past President of the International Neural Network Society, and Past Editor in Chief of the Transactions of Biomedical Engineering, as well as a former member of the Advisory Science Board of the FDA. He holds 5 patents and has submitted seven more. Dr. Principe was supervisory committee chair of 65 Ph.D. and 67 Master students, and he is an author of more than 500 refereed publications (3 books, 4 edited books, 14 book chapters, 200 journal papers and 380 conference proceedings).