A Review of *Minimum Classification Error* Training

Yaakov Stein

15 Jan. 1994

Abstract

Shigeru Katagiri and various co-authors have (re)introduced a nonstandard error measure which can be used in training almost any classifier. This measure is usually called *MCE* for Minimum Classification Error, and is often coupled with an equally nonstandard technique, known as *GPD* for Generalized Probabilistic Descent, for updating the classifier's parameters due to the error. In the present paper, we will present this measure and technique, and describe its use in speech recognition applications.

1 Introduction

We will deal with decision theoretic classifiers, which input a point $\mathbf{x}\epsilon\mathcal{R}^n$, and output a classification C_k where $k\epsilon\{1\ldots m\}$. We will assume that the *architecture* of the classifier is fixed, but that the mapping from the input to the classification depends on a vector of parameters $\mathbf{W}$. In such cases the *selector representation* is most often employed, as depicted in figure 1. In this representation each possible classification is assigned an output, and the class corresponding to the output unit with highest value is considered the winner. Although such a 'winner-takes-all' strategy is actually sufficient to utilize such classifiers (and this is the gist of the MCE formalism), we will see in the next section that it is prevalent to desire the winner to be unity for inputs which indeed belong to the class, and zero otherwise. Thus one is led to interpret the outputs as probabilities. Other representations are possible, eg. binary encoding of the class index, but this format allows one to hope for a probabilistic-like interpretation of the output values.

Supervised learning is a generic process whereby a parametric classifier can be optimized for a particular problem based on the existence of a *teacher*, who somehow knows how to perform the desired classification. For example in the *learning by examples* paradigm the teacher prepares a finite set of classified examples (the *training set*), which can be studied by the *student* classifier. In alternative paradigms, an *oracle* may exist which can be consulted for any input the student desires. In any case the goal of training is to minimize the probability of classification error on *any possible* input, not only on inputs which the student has seen during the training process. In order to estimate the probability of such errors, a distinct *test set* is commonly used.

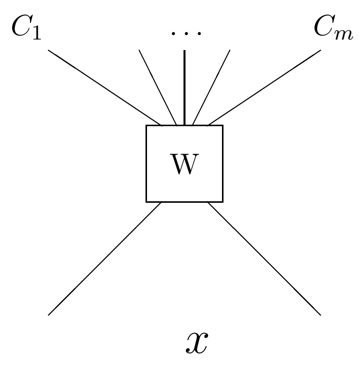

Figure 1: The selector representation

The training itself consists of updating the parameter vector $\mathbf{W}$, in such a way that the classification error (hopefully) decreases. We will not dwell for the time being on the exact algorithm used in updating the parameters, rather we will concentrate on the method of quantifying the error.

2 Cost Functions

We wish now to sharpen the concept of classifier optimization introduced in the previous section. We assume that we are given a classifier architecture and that all of the free parameters of the classifier are specified by the vector $\mathbf{W}$. The task at hand is to define a *cost function* $L(\mathbf{W})$ which characterizes this classifier's 'quality'. The ideal classifier would have zero cost, while increasing cost corresponds to decreasing classification success.

We would certainly like to use the number of classification errors (or equivalently the probability of misclassification) made on the training set (or even better on the *test* set) as the cost function to be minimized. However, this method of quantifying the classifier error has serious disadvantages. To see this, consider the behavior of the number of misclassifications as a function of the parameter vector $\mathbf{W}$. Perturbing $\mathbf{W}$ will cause the output of the classifier to vary, but for most $\mathbf{W}$ will not induce changes in the actual classification. For example, for the selector representation the classification is determined by the output unit with the highest activity, and only parameter perturbations which cause another output unit to pass the previous winner will cause the actual classification to change. We thus see that in the space of parameters, the number of classification errors is a piecewise-continuous function.

Most methods of classifier parameter optimization are *iterative descent* methods. Such methods search for a minimum in the cost function as a function of the parameter vector

$\mathbf{W}$ by starting with some initial guess $\mathbf{W}_0$ and repeatedly updating by

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \delta \mathbf{W}_t \tag{1}$$

such that

$$L(\mathbf{W}_{t+1}) < L(\mathbf{W}_t) \ . \tag{2}$$

The simplest example is of course *gradient descent* which choses to 'slide down' the local gradient of the cost function in w dimensional parameter space by a constant amount

$$\delta \mathbf{W}_t = -\varepsilon \nabla L(\mathbf{W}) \tag{3}$$

where the gradient operator is defined as

$$\nabla \equiv (\frac{\partial}{\partial W_1}, \frac{\partial}{\partial W_2}, \ldots, \frac{\partial}{\partial W_w}) \ . \tag{4}$$

More complex methods, such as *conjugate gradient*, employ more elaborate heuristics to set the step size, but still require the cost function to be a continuously differentiable function of the parameters. The derivative of number of classification errors being zero almost everywhere and infinite where it is not, is thus an extremely poor choice for iterative descent.

We are thus lead to consider alternative cost functions. The requirements for such a function are

- $L(\mathbf{W}) = 0$ if and only if a *perfect* solution is found,
- $L(\mathbf{W}_1) < L(\mathbf{W}_2)$ indicates that $\mathbf{W}_1$ is a better solution,
- $L(\mathbf{W})$ is continuously differentiable.

The most common technique for defining such a cost function involves assuming concrete values for the output of the selector representation classifier. We mentioned in the previous section the probability-like choice, wherein we require the output unit corresponding to the correct class to be unity and all others to be zero. A second useful choice is $+1$ for the correct class and -1 for the others. We will refer to the former as the $\{0, 1\}$ representation and to the latter as the $\{\pm 1\}$ one. Once concrete values have been assigned, the desired output is a vector $\mathbf{D} = (D_1, D_2, \ldots D_n)$ (with D_i in $\{0, 1\}$ or $\{\pm 1\}$ respectively), while the actual output of the classifier is a vector $\mathbf{O}(\mathbf{W}) \epsilon \Re^n$. Thus one may build cost functions based on measures of distance between vectors in $\Re^n$, as long as the metric obeys the above requirements. We will define the *loss* of an input pattern as the distance between the desired output and the actual output. The overall cost function will be the sum of the losses for all the input patterns under consideration.

The most common metric employed as a loss function is the squared error

$$\sum_{i=1}^{n} \Big(O_i(\mathbf{W}) - D_i \Big)^2 \tag{5}$$

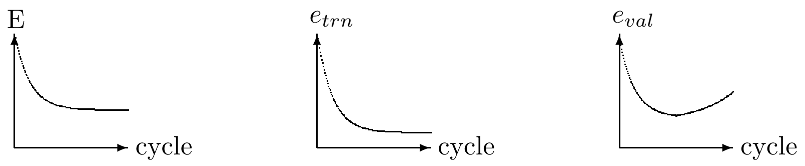

Figure 2: The three quantities to be observed during training

which, due to an analogy with physical systems, is often called the *energy.* Another metric [14], employed when both O_i and D_i are constrained to the interval $[0 \ldots 1]$, is the Kullback-Liebler divergence

$$\sum_{i=1}^{n} \left(D_i \ln \frac{D_i}{O_i} + (1 - D_i) \ln \frac{1 - D_i}{1 - O_i} \right) \tag{6}$$

which, due to a similar analogy is also called the *entropy.* Both of these functions have global minima for $O_i = D_i$, and the choice between them is application dependent.

When using such distance measures for training, it is common practice to monitor three quantities as training proceeds (see figure 2). First, the cost function itself, as defined on the training set (ie. the sum of the losses for all training patterns). Second, the number of errors the classifier makes on the training set. Lastly, the number of errors made on a *validation set*, which is a set of classified examples which is not used for training. The classification performance on the validation set is meant to reflect the generalization to be expected on arbitrary inputs. The validation set is often used as an indication of when to terminate training, since increase in validation error while training error is still decreasing is attributable to overtraining.

2.1 The Probabilistic Interpretation of Least Squared Error

Let us deal with the one-output (two-class) case. We wish to determine the meaning of the minimization of the squared error in probabilistic terms. We will show that for the one-output case, minimizing the squared error of the output leads us to a classifier which is the minimum mean square error approximation to the Bayes optimal discriminant [6, 13].

Let us denote two classes by C_a and C_b with associated outputs a and b, where $(a, b) = (0, 1)$ or $(a, b) = (-1, +1)$. Without limiting generality we order the training set of N examples such that the first N_a patterns belong to C_a and the rest to C_b. We further denote the output of the classifier with parameters $\mathbf{W}$ when presented with input $\mathbf{x}$ by $O(\mathbf{x}; \mathbf{W})$. Then the squared error cost function can be written

$$E(\mathbf{W}) = \frac{1}{N} \left\{ \sum_{x=1}^{N_a} [O(\mathbf{x}; \mathbf{W}) - a]^2 + \sum_{x=N_a+1}^{N} [O(\mathbf{x}; \mathbf{W}) - b]^2 \right\} . \tag{7}$$

We now assume that the training set is large $N \to \infty$ so that the a priori class probabilities are well approximated by $\frac{N_a}{N} \to p(C_a)$ and $\frac{N_b}{N} \to p(C_b)$ and thus by the strong law of large numbers

$$\begin{aligned} E(\mathbf{W}) &\rightarrow \int [O(\mathbf{x};\mathbf{W}) - a]^2 p(x,C_a)dx + \int [O(\mathbf{x};\mathbf{W}) - b]^2 p(x,C_b)dx \\ &= \int O^2(\mathbf{x};\mathbf{W})[p(x,C_a) + p(x,C_b)]dx - 2\int O(\mathbf{x};\mathbf{W})[ap(x,C_a) + bp(x,C_b)]dx \qquad (8) \\ &\quad + a^2 \int p(x,C_a)dx + b^2 \int p(x,C_b)dx \qquad (9) \end{aligned}$$

where the joint probability $p(x, C_i)$ obeys

$$\begin{aligned} p(x,C_i) &= p(C_i|x)p(x) = p(x|C_i)p(C_i) \quad \text{(Bayes Theorem)} \\ \int p(x,C_i)dx &= p(C_i) \\ p(x,C_a) + p(x,C_b) &= p(x)\ . \end{aligned}$$

Defining

$$\begin{aligned} g(x) &= \frac{ap(x,C_a) + bp(x,C_b)}{p(x)} = ap(C_a|x) + bp(C_b|x) \\ &= p(C_1|x) \qquad \text{for } (a,b) = (0,1) \\ &= p(C_1|x) - p(C_{-1}|x) \qquad \text{for } (a,b) = (-1,+1) \end{aligned}$$

which for the $(0,1)$ case is the a posteriori probability of C_1 and for the $(-1,+1)$ case the Bayes optimal discriminant, we find

$$\begin{aligned} E(\mathbf{W}) &= \int [O^2(\mathbf{x};\mathbf{W})p(x) - 2O(\mathbf{x};\mathbf{W})g(x)p(x)]dx + a^2p(C_a) + b^2p(C_b) \\ &= \int [O(\mathbf{x};\mathbf{W}) - g(x)]^2 - \int g^2(x)p(x)dx + a^2p(C_a) + b^2p(C_b)\ . \end{aligned}$$

Since only the first term above depends on $\mathbf{W}$, minimizing $E(\mathbf{W})$ by varying $\mathbf{W}$ is equivalent to searching for a classifier function $O(\mathbf{x};\mathbf{W})$ which best approximates $g(x)$. We have thus proven that for the one-output case, minimal energy training leads to output values which best approximate the a posteriori probability. This is indeed surprising, since only the two extreme values a and b are presented during training! Of course the usual caveats to approximation methods apply, eg. the rule may not be realizable in terms of the classifier architecture, there may be local minima, there may be insufficient training data, etc.

For the case of $(a,b) = (0,1)$, and for classifiers which employ all positive sigmoidal functions at the output (for example MLPs), one can say even more. Let us call the output *before* the sigmoidal squashing $Z(\mathbf{x};\mathbf{W})$, so that

$$O(\mathbf{x};\mathbf{W}) = \frac{1}{1 + e^{-Z(\mathbf{x};\mathbf{W})}} \qquad \text{or} \qquad Z(\mathbf{x};\mathbf{W}) = \ln \frac{O(\mathbf{x};\mathbf{W})}{1 - O(\mathbf{x};\mathbf{W})}\ .$$

We have seen that $O(\mathbf{x};\mathbf{W}) \approx p(C_1|\mathbf{x})$ and so (noting that the normalization requires $1-p(C_1|\mathbf{x}) = p(C_0|x)$)

$$Z(\mathbf{x};\mathbf{W}) = \ln\frac{p(C_1|\mathbf{x})}{p(C_0|\mathbf{x})} = \ln\frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_0)} + \ln\frac{p(C_1)}{p(C_0)}\ . \tag{10}$$

The fact that minimization of squared error of the squashed output leads to a type of maximum likelihood criterion on the unsquashed output should not surprise us. Along the boundary between two classes the two posterior class probabilities must be equal, and thus a function which is zero along this boundary and increases monotonically when approaching one of the classes can always be considered a 'distance' to the boundary.

If MLP training techniques induce a maximum likelihood criterion, such as is used in HMM training, what then are the advantages of the neural network approach? Firstly, a (sufficiently large) neural network is a *universal approximator* and thus, in principle, one need not deal with the effects of model mismatch. Secondly, it has been shown that neural networks are very efficient in their use of input information, often surpassing the universal Vapnik-Chervonenkis upper bound for generalization error as a function of training set size. Thirdly, neural networks are flexible: one can easily add false alarms to the training set, combine neural modules, and add auxilliary information even without theoretical understanding of its relevance.

Now let us consider what happens when there are multiple outputs. Limiting ourselves to the $(a,b) = (0,1)$ representation, we can show in a way analogous to the above, that minimization of the energy leads to the minimization of

$$\sum_{i=1}^{m}\int[O(\mathbf{x};\mathbf{W}) - p(C_i|\mathbf{x})]^2 p(x)dx$$

ie. to the simultaneous approximation of the m posterior class probabilities $p(C_i|\mathbf{x})$ according to the 'minimal average-of-squared-errors' criterion. However, $\sum_{i=1}^{m} O(\mathbf{x};\mathbf{W}) \neq 1$ and thus the individual outputs are not really probabilities at all. One can simply normalize after training, but this does not really ameliorate the situation since this is no longer a least squares solution to the original problem. There are several possible solutions to this problem, including adding a special normalization layer (involving much more elaborate training) and building histograms or Parzen windows on the classifier outputs.

The maximum entropy criterion can be discussed in a fashion similar to the above discussion for the energy. The results are that maximum entropy is equivalent to the maximization of the mutual information between the class identification and the input

$$I(C_i;\mathbf{x}) = H(C_i) - H(C_i|\mathbf{x})$$

where the a priori uncertainty is

$$H(C_i) = -\sum p(C_i)\ln p(C_i)$$

and the *equivocation* or uncertainty as to the class identification *after* observing $\mathbf{x}$ is

$$H(C_i|\mathbf{x}) = -\int p(\mathbf{x})[\sum p(C_i|\mathbf{x}) \ln p(C_i|\mathbf{x})]dx \ .$$

Note that if $\mathbf{x}$ gives no information as to the proper classification then $H(C_i|\mathbf{x}) = H(C_i)$ and $I(C_i;\mathbf{x}) = 0$; while if the classification is obvious $H(C_i|\mathbf{x}) = 0$ and then $I(C_i;\mathbf{x}) = H(C_i)$.

2.2 The Problem with the Least Squared Error Criterion

We have seen that the minimization of energy is both simple to perform and has a reasonable probabilistic interpretation. Energy minimization training along with monitoring the three quantities of figure 2 should thus lead us to a high quality solution. There is still, however, a serious mismatch between the desired criterion of minimal error probability and the criterion actually used.

To see this we present somewhat pathological cases. We do not mean to imply that only for such unusual cases is the mismatch of importance – one may obtain suboptimal error performance in more general settings as well. The idea of this section is simply to demonstrate the mismatch by reductio ad absurdum.

First let us note that while zero energy indeed implies perfect classification, the converse is not true. It is quite possible (and even commonplace) for the classification error to be zero, while the energy is positive. This occurs simply because a proper classification occurs when the appropriate unit is maximal, even if it is not +1 and all the others inactive. Thus the energy, or indeed any criterion based on concrete output values, does not supply a good indication of when training is unnecessary, and thus unwise overtraining may be performed.

Secondly we observe that the energy can decrease without a complimentary decrease in classification error – indeed the error may even *increase*. For example consider an input which belongs to class number 1 and for which the output unit activations are

$$\lambda + \varepsilon, \lambda - \varepsilon, \varepsilon, \ldots, \varepsilon \qquad\qquad (m - 2 \ \epsilon\text{s}) \ .$$

This clearly correctly classifies the input, and the energy is easily seen to be $E = 1 + 2\lambda^2 - 2\lambda - 2\varepsilon + m\varepsilon^2$. Now assume that after a particularly unfortunate training step the output layer becomes

$$\lambda - \varepsilon, \lambda + \varepsilon, \eta\varepsilon, \ldots, \eta\varepsilon$$

which misclassifies the input, but has energy $E = 1+\lambda^2-2\lambda+2\varepsilon+m\eta^2\varepsilon^2+2(1-\eta^2)\varepsilon^2$ which will be lower than the original energy if $(1-\eta^2) > \frac{4}{(C-2)\varepsilon}$ (always true for a sufficiently large m). For this case the energy has indeed decreased, but the classification has become *worse*!

3 Defining the MCE Cost Function

The message of the MCE approach is that, despite the objections raised in the beginning of section 2, one *can* use the criterion of decreasing classification errors and thus avoid the pathological situations brought about by energy minimization. The idea is to define a cost function which is directly related to classification error, but at the same time is continuously differentiable.

In order to understand the MCE cost function, it is advantageous to divide its definition into four stages. The first stage is simply the basic discriminant function, and is thus architecture and problem dependent. This discriminant is the only function needed for actually *classifying* an input – all the rest are for training purposes only. The last stage is the cost function itself, which functions as the classifier error to be minimized by the training process. The second and third stages are auxilliary functions, which could equally well be absorbed into the cost function, but are defined separately for purely didactic purposes.

3.1 The Discriminant Function

We will denote this function by $g_k(\mathbf{x}; \mathbf{W})$, and its meaning is as follows. Given a classifier specified by the parameter vector $\mathbf{W}$, to what degree does the input $\mathbf{x}$ belong to class C_k? The discriminant function is always assumed to be nonnegative, but in the MCE literature there are two contradictory types of discriminant; for one type the discriminant is maximal for the desired class, while for the other it is minimal. The origin of this discrepancy is simple to understand. For DTW-based speech recognizers it is convenient to use the dynamic programming distortion score as the discriminant, and thus we desire the discriminant to be minimal. On the other hand, were the a posteriori probabilities known, it would be natural to identify $g_k(\mathbf{x}) = p(C_k|\mathbf{x})$, and to desire maximal discriminant.

The discriminant function does not depend on the fact that we intend using MCE in its training, and is chosen according to the task at hand and the architecture of the classifier chosen. As was mentioned before, the discriminant function is also the only function which exists *after* the training, and thus its determination is the task to be accomplished. Alternative training philosophies which share a given discriminant function will in general produce different discriminant functions (ie. different parameter vectors $\mathbf{W}$).

3.2 The Misclassification Measure

This function, which we shall denote $d_k(\mathbf{x}; \mathbf{W})$, indicates to what extent an input in the training set $\mathbf{x}$ belongs to class C_k *as compared to the other classes* C_j. Here we assume that k is the class to which $\mathbf{x}$ *really* belongs, which is known for all training set examples. Since this is a **mis**classification measure, we desire $d_k(\mathbf{x}; \mathbf{W}) > 0$ to signify that the input $\mathbf{x}$ is *not* properly classified. Conversely, for proper classification $d_k(\mathbf{x}; \mathbf{W}) < 0$ and thus the training procedure must strive to minimize $d_k(\mathbf{x}; \mathbf{W})$.

Actually, the theoretical MCE literature defines a family of misclassfication measures, specified by a parameter which weights the influence of the incorrect classes. The exact form proposed, for discriminants maximal for the desired class, is

$$d_k(\mathbf{x};\mathbf{W}) = -g_k(\mathbf{x};\mathbf{W}) + \left[\frac{1}{m-1}\sum_{j\neq k} g_j(\mathbf{x};\mathbf{W})^{\eta}\right]^{\frac{1}{\eta}} \tag{11}$$

where m is the number of classes, and $\eta > 0$ is the aforementioned weighting parameter. Note that the discriminant is essentially being enumerated, with the proper class being taken with opposite sign. The particular form of enumeration (the so called L_p or *Hölder-norm*) is somewhat peculiar, and gives the characteristic formula structure which one observes in all MCE articles. The basic idea behind this enumeration is quite simple. For $\eta = 1$ the second term on the rhs of (11) is the simple average of the discriminant for all the incorrect classes. For $\eta = 2$ we have the Pythagorean average $\sqrt{g_2^2 + g_3^2 + \cdots}$, and for $\eta \to \infty$ only the incorrect class with the highest discriminant remains. In this limit, denoting the index of the best matching incorrect class by i, we have

$$d_k(\mathbf{x};\mathbf{W}) = g_i(\mathbf{x};\mathbf{W}) - g_k(\mathbf{x};\mathbf{W}) \tag{12}$$

which is the form most often used in actual applications, and is similar to the criterion of the Neyman-Pearson test. For the special and idealized case of two classes $k = 1, 2$ and exact a posteriori probabilities $g_k(\mathbf{x}) = p(C_k|\mathbf{x})$, we obtain in this limit,

$$d_1(\mathbf{x}) = -\Big(p(C_1|\mathbf{x}) - p(C_2|\mathbf{x})\Big) \tag{13}$$

which is the negative of the Bayes optimal discriminant for this case, with the MAP decision rule corresponding to

$$d_1(\mathbf{x}) < 0 \Rightarrow \mathbf{x}\epsilon C_1 \qquad \text{and} \qquad d_1(\mathbf{x}) > 0 \Rightarrow \mathbf{x}\epsilon C_2\ . \tag{14}$$

For $\eta < \infty$ the MCE misclassification measure does take classes other than the most significant into consideration.

For discriminant functions of the type which are minimal for the desired class, equation (11) must be replaced with

$$d_k(\mathbf{x};\mathbf{W}) = -g_k(\mathbf{x};\mathbf{W}) + \left[\frac{1}{m-1}\sum_{j\neq k} g_j(\mathbf{x};\mathbf{W})^{-\eta}\right]^{-\frac{1}{\eta}} \tag{15}$$

in order that the $\eta \to \infty$ limit select the *best* matching incorrect class.

3.3 The Loss Function

In order to count the number of errors a classifier makes, we *should* calculate

$$N_{errors} = \sum_{\mathbf{x}} \Theta \left(d_k(\mathbf{x}; \mathbf{W}) \right) \tag{16}$$

where $\Theta(z)$ is the step function ($\Theta(z) = 0$ for $z < 0$ and $\Theta(z) = 1$ for $z > 0$) and the sum is over all inputs in the set under consideration, usually the training set. In this fashion we could directly count the number of misclassifications, and were one to attempt to employ this quantity as a cost function, one would be lead to the problems mentioned in section 2 supra. Instead we wish to smooth the step functions, so that the cost function's derivatives will be everywhere continuous and nonzero. However, this smoothing must be performed without sacrificing the fundamental characteristics of the step function, namely by using in its stead a sigmoid function $\sigma(z)$ obeying

- monotonicity $z_1 < z_2 \Rightarrow \sigma(z_1) < \sigma(z_2)$,
- bounded $0 < \sigma(z) < 1$ everywhere,
- unbiased $\sigma(0) = \frac{1}{2}$, and
- proper limits $\sigma(z) \xrightarrow{z \to -\infty} 0$ and $\sigma(z) \xrightarrow{z \to \infty} 1$.

The particular sigmoid usually employed is the *logistic* or *Fermi* sigmoid

$$\sigma(z) \equiv \frac{1}{1 + e^{-\beta z}} \ . \tag{17}$$

Using the sigmoid smoothing, we define the loss function

$$\ell_k(\mathbf{x}; \mathbf{W}) = \sigma \left(d_k(\mathbf{x}; \mathbf{W}) \right) \tag{18}$$

and indeed note that when d_k becomes large and negative (indicating proper classification) the loss approaches zero; while a definite misclassification (one for which $d_k \to \infty$) causes a loss of unity. Thus, at least for large β, summing over $\ell_k(\mathbf{x}; \mathbf{W})$ should give a good approximation to the quantity in (16).

Now that we have smoothed the enumeration of the number of errors, won't we return to a cost function which is similar in its faults to squared error? The answer is *no*, for several related reasons. One can in principle take the limit $\beta \to \infty$ and approach the step function enumeration as closely as desired. In addition, even for relatively small β, far from transitions the loss function approximates the step function. This behavior should be contrasted with that of a squared distance function which increases without limit rather than saturating.

3.4 The Cost Functions

We are finally ready to define the overall cost functions. The literature deals with two types of overall costs, the empirical average cost and the expected cost. The former (**MCE**) can be minimized using standard gradient descent methods, while the latter requires a technique known as Generalized Probabilistic Descent (**GPD**).

The empirical average cost function is simply the average of the loss over the entire *training* set

$$L_0(\mathbf{W}) = \frac{1}{N} \sum_i \ell_{k_i}(\mathbf{x}_i; \mathbf{W}) \tag{19}$$

where $i = 1 \dots N_t$ runs over all examples in the training set, $\mathbf{x}_i$ is the training set example itself, and k_i stands for the index of the class to which $\mathbf{x}_i$ really belongs. In contradistinction to this restricted definition, the expected cost function (the GPD cost) is defined to be

$$L(\mathbf{W}) = \left\langle \ell_{k_x}(\mathbf{x}; \mathbf{W}) \right\rangle \tag{20}$$

where here the average is taken over all possible inputs $\mathbf{x}$ (not just the training set examples), and k_x stands for the index of the class to which $\mathbf{x}$ really belongs (which in general is unknown).

How do we utilize these cost functions? We want to find a parameter vector $\mathbf{W}$ to be used in $g_k(\mathbf{x}; \mathbf{W})$ as discussed in subsection 3.1 above, optimal in the sense that the cost function will be minimized. We *directly* minimize the cost function by iteratively updating the parameter vector $\mathbf{W}$ via equation (1). We emphasized the word 'directly' since unlike the alternative methods, we do not define desired outputs (as in least square error) and make no assumptions about the probability distributions (as in maximum likelihood).

Let us deal first with the MCE cost function, and employ simple gradient descent (3) as the minimization technique. We will now explicitly show that gradient descent reduces the MCE cost function for small enough step size ε. While this proof is well known, and indeed trivial, we include it here for comparison with the GPD derivation to be given below. For small step size, $L_0(\mathbf{W}_{t+1})$ can be expanded to first order as follows

$$\begin{aligned} L_0(\mathbf{W}_{t+1}) &= L_0(\mathbf{W}_t + \delta\mathbf{W}_t) \\ &\approx L_0(\mathbf{W}_t) + \delta\mathbf{W}_t \cdot \nabla L_0(\mathbf{W}_t) \\ &= L_0(\mathbf{W}_t) - \varepsilon |\nabla L_0(\mathbf{W}_t)|^2 < L_0(\mathbf{W}_t) \end{aligned} \tag{21}$$

as required. Using the definition (19) in the expression for gradient descent (3), we can write explicitly

$$\delta\mathbf{W}_t = -\frac{\varepsilon}{N} \sum_i \nabla \ell_{k_i}(\mathbf{x}_i; \mathbf{W}_t) \tag{22}$$

and furthermore continue to differentiate through the sigmoid, (etc. etc.) until we arrive at the derivative of the discriminant itself.

In the late sixties Shunichi Amari developed a theory of *adaptive pattern classifiers*, ie. classifiers which dynamically adjust themselves to changing environments. Such classifiers need not be pretrained and then used statically, rather the training is kept 'on-line' and thus can follow dynamically varying probability distributions of the classes. Amari concentrated on *linear* classifiers, as these were the state-of-the-art of the time, but generalized the theory to the general case as well. The main contribution of this work is the development of an adaptive algorithm (later to be called GPD) which provably converges to a locally optimal classifier in a probabilistic sense. Since Amari dealt with input patterns that are constantly arriving rather than in a fixed training set, the optimality is from the point of view of *any* input $\mathbf{x}$ chosen according to the probability distribution used to generate the examples previously seen. This can be applied to the case of a defined training set as well, in which case the training examples are repeatedly presented, but we strive to reduce the expected cost (20).

Let us write equation (20) more explicitly.

$$L(\mathbf{W}) = \sum_k P(C_k) \int \ell_k(\mathbf{x};\mathbf{W}) p(\mathbf{x}|C_k) dx \tag{23}$$

where the sum is over all the classes, the integral over all inputs in class C_k, $P(C_k)$ is the a priori probability of the class, and $p(\mathbf{x}|C_k)$ is the probability distribution of the class. Once again we will update the parameters by equation (1), but now we wish to have $L(\mathbf{W}_{t+1}) < L(\mathbf{W}_t)$. We now proceed as in (21)

$$\begin{aligned} L(\mathbf{W}_{t+1}) &= L(\mathbf{W}_t + \delta\mathbf{W}_t) \\ &\approx L(\mathbf{W}_t) + \delta\mathbf{W}_t \cdot \nabla L(\mathbf{W}_t) \end{aligned} \tag{24}$$

but since the probabilities $P(C_k)$ and $p(\mathbf{x}|C_k)$ are not known, we can not force $\delta\mathbf{W}_t = \varepsilon\nabla L(\mathbf{W}_t)$. We *can* obtain reduction in $L(\mathbf{W}_t)$ *on the average* (thus **Probabilistic** Descent) by requiring [1]

$$\left\langle \delta\mathbf{W}_t \cdot \nabla L(\mathbf{W}_t) \right\rangle \leq 0 \tag{25}$$

which leads to an update rule which takes only the present input $\mathbf{x}_t$ into account

$$\delta\mathbf{W}_t = -\varepsilon \mathbf{U} \nabla \ell_k(\mathbf{x}_t;\mathbf{W}_t) \tag{26}$$

where $\mathbf{U}$ is a positive definite matrix. To show that this indeed leads to the desired reduction in expected cost, we need only to calculate

$$\begin{aligned} \left\langle \delta\mathbf{W}_t \cdot \nabla L(\mathbf{W}_t) \right\rangle &= \left\langle \delta\mathbf{W}_t \right\rangle \cdot \nabla L(\mathbf{W}_t) \\ &= -\varepsilon \nabla L(\mathbf{W}_t)\mathbf{U} \left\langle \nabla \ell_k(\mathbf{x}_t;\mathbf{W}_t) \right\rangle \\ &= -\varepsilon \nabla L(\mathbf{W}_t)\mathbf{U}\nabla \left\langle \ell_k(\mathbf{x}_t;\mathbf{W}_t) \right\rangle \\ &= -\varepsilon \nabla L(\mathbf{W}_t)\mathbf{U}\nabla L(\mathbf{W}_t) \leq 0 \end{aligned} \tag{27}$$

the inequality following from the positive-definiteness of $\mathbf{U}$, and equality only being possible if $\nabla L(\mathbf{W}_t) = 0$ ie. a local optimum is found.

One can further show that if the GPD training procedure is supplied with an infinite sequence of inputs $\mathbf{x}$ and the step size is varied according to any schedule which obeys

$$\sum_{t=1}^{\infty} \varepsilon_t = \infty \tag{28}$$

and

$$\sum_{t=1}^{\infty} \varepsilon_t^2 < \infty$$

then $\mathbf{W}_t$ converges with probability one to a local minimum of $L(\mathbf{W})$. The convergence can be shown to be extremely efficient (in fact exponential); however the choice of ε induces a trade-off between the speed and the accuracy of convergence.

Since $\mathbf{U}$ may be taken to be the identity matrix (which is certainly positive definite), it seems that equation (26) and (22) are essentially identical. The basic differences are in the step size ε which is constant for MCE but must taken according to (28) for GPD, and in the added degrees of freedom of $\mathbf{U}$. In applications where $\mathbf{U}$ is chosen wisely and varied (as ε_t is) it is called $\mathbf{U}_t$ the *learning matrix.*

4 Using MCE and GPD in Specific Architectures

After the somewhat lengthy theoretical discussion of the previous sections, we turn now to the practical application of the principles of MCE/GPD training as applied to popular classifier architectures. One of the strengths of the MCE methodology is its wide applicability – indeed almost any parametric classifier can be trained via MCE, and it turns out that the required modifications to existing training algorithms are usually minor. In the following we will discuss the application of MCE to three classifier structures, multilayer perceptron neural networks (MLP), hidden Markov models (HMM) and dynamic time warping (DTW). MCE has been applied to other structures as well (eg. LVQ) , and even to the design of feature extractors [2].

4.1 MLP

It is perhaps simplest to begin with the application of the MCE/GPD technique to feed-forward neural network classifiers. These classifiers consist of a number of processing elements called 'neurons', n of which are designated inputs, and m of which are outputs. The input neurons may feed intermediate (hidden) neurons which in turn may feed still other neurons, and so on until finally the output neurons are reached. No loops (feedback connections) are allowed. A pattern is presented by copying its values into the input neurons and forward propagating the signal towards the output. Training patterns are supplied with desired outputs and the actual output can be compared with these. The usual training technique consists of minimizing the squared error criterion by backwards propagation of the error from the output layer back towards the input.

Let us denote the output of neuron i by z_i. For all neurons which are not input neurons

$$z_i = \hat{\sigma}(y_i) = \hat{\sigma}(\sum_j w_{ij} z_j) \tag{29}$$

where $y_i = \sum_j w_{ij} z_j$ is the neuron's *potential*, the sum is over all neurons which feed neuron i, w_{ij} is the weight connecting neuron j to neuron i (any bias terms have been absorbed into the sum by assuming a constant unit), and the function $\hat{\sigma}(y)$ is a monotonic 'squashing' function, usually sigmoidal in character, which transforms the neuron's potential y_i into its output z_i. We use $\hat{\sigma}(y)$ to denote tanh-like sigmoids, which tend to -1 for large negative arguments, saving $\sigma(y)$ for all-positive sigmoids, as in equation (17). The squared pattern error (5) is

$$E_{sq.err} = \sum_i (z_i - D_i)^2 \tag{30}$$

where the sum is over the m output neurons only, and D_i are the desired output values. Gradient descent implies changing the weights feeding output units i as follows (neglecting momentum terms) :

$$\delta w_{ij} = -\varepsilon \frac{\partial E_{sq.err}}{\partial w_{ij}} \tag{31}$$

where the derivative can be calculated by the chain rule

$$\frac{\partial E_{sq.err}}{\partial w_{ij}} = \frac{\partial E_{sq.err}}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}} = \frac{\partial E_{sq.err}}{\partial y_i} z_j$$

and furthermore

$$\frac{\partial E_{sq.err}}{\partial y_i} = \frac{\partial E_{sq.err}}{\partial z_i} \frac{dz_i}{dy_i} = \frac{\partial E_{sq.err}}{\partial z_i} \hat{\sigma}'(y_i) \tag{32}$$

where $\hat{\sigma}'(y)$ is the derivative of the squashing function, and finally

$$\frac{\partial E_{sq.err}}{\partial z_i} = 2(z_i - D_i) \ . \tag{33}$$

Thus the weight change for weights feeding an output is given (absorbing the constant factor of 2 into ε) by

$$\delta w_{ij} = -\varepsilon z_j \hat{\sigma}'(y_i)(z_i - D_i) \ . \tag{34}$$

For non-output neurons, the expression is somewhat more complex, but will not be required here.

Let us now substitute the MCE error criterion for the least squared one. Formally, we replace the above squared error energy with the loss function $E_{MCE} = \ell_k(\mathbf{x}; \mathbf{W}) = \sigma(d_k(\mathbf{x}; \mathbf{W}))$ which in turn is derived from the following misclassification measure

$$d_k(\mathbf{x}) = -z_k + \left[\frac{1}{m-1} \sum_{j \neq k} z_j^{\eta} \right]^{\frac{1}{\eta}} \overset{\eta \to \infty}{\longrightarrow} z_h - z_k \tag{35}$$

where h signifies the output $i \neq k$ with maximal activation. It is obvious that use of this cost function rather than the squared error will only change the expressions for updating weights which feed output neurons. Indeed, the only change to be made in the above derivation is in the derivative with respect to z_i; equation (33) being replaced by

$$\frac{\partial E_{MCE}}{\partial z_i} = \frac{\partial \ell_k(\mathbf{x};\mathbf{W})}{\partial z_i} = \sigma'\left(d_k(\mathbf{x})\right)\frac{\partial d_k(\mathbf{x})}{\partial z_i} \tag{36}$$

with $d_k(\mathbf{x})$ given above. Explicitly differentiating

$$\frac{\partial d_k(\mathbf{x})}{\partial z_i} = \begin{cases} -1 & i = k \\ \frac{z_i^{\eta-1}}{m-1}\left[\frac{1}{m-1}\sum_{j\neq k} z_j^{\eta}\right]^{\frac{1}{\eta}-1} & i \neq k \end{cases} \tag{37}$$

$$\overset{\eta\to\infty}{\longrightarrow} \begin{cases} -1 & i = k \\ +1 & i = \underset{j\neq k}{\operatorname{argmax}}\ z_j \\ 0 & \text{else} \end{cases} \tag{38}$$

so that we finally arrive at the correction to the weight connecting neuron j to output neuron i

$$\begin{aligned} \delta w_{ij} &= -\varepsilon\frac{\partial E_{MCE}}{\partial w_{ij}} = -\varepsilon z_j \hat{\sigma}'(y_i)\sigma'\left(d_k(\mathbf{x})\right)\frac{\partial d_k(\mathbf{x})}{\partial y_i} \\ &\overset{\eta\to\infty}{\longrightarrow} \varepsilon z_j \hat{\sigma}'(y_i)\sigma'\left(d_k(\mathbf{x})\right)\begin{cases} +1 & i = k \\ -1 & i = \underset{j\neq k}{\operatorname{argmax}}\ y_j \\ 0 & \text{else} \end{cases} . \end{aligned} \tag{39}$$

The meaning of the corrections introduced by MCE is now made clear by comparing equations (34) and (39). In conventional backprop the contribution of an output neuron i to the squared error as given by equation (33), is the difference between the activation and the desired value, and this is what is backpropagated. Since for MCE there is no *desired* value, we backpropagate in its stead the expression in equation (36), which is the product of two factors. For large η, the factor given in equation (37) causes the desired output to be strengthened (ie. d_k becomes more negative), the highest undesired output to be weakened, and no other weights are updated. These corrections are further multiplied by a symmetric weighting function, which for the standard sigmoid (equation (17) with $\beta = 1$) is $\sigma'(d_k) = \sigma(d_k)\left(1 - \sigma(d_k)\right)$. This factor is bounded $0 < \sigma'(d_k) \leq \frac{1}{4}$ and takes its maximum value when $d_k = 0$ ie. when the pattern is close to an intercategory boundary. Thus when patterns are close to borders (are only marginally classified) the parameter vector is changed by large steps, while for patterns far from these borders (including both correctly and incorrectly classified patterns) the parameters vector changes are smaller.

In [?] the MCE-MLP is tried on several benchmark classification tasks. The first task was an artificially generated two class Gaussian mixture in two dimensions. Three kinds of *linear* classifiers were trained, according to the least squared error, perceptron, and MCE criteria, and the test set error percentages (averaged over four runs) were 14.91%, 11.18% and 9.85%, respectively. The second task was Fisher's well known 'Iris' database, which consists of four measurements on three types of Iris flowers. Here both linear classifiers and three-layer MLPs were trained, with the linear classifiers attaining 15.1%, 14%, and 4% error, the least square MLP 12.3% error, and the MCE-MLP being the best with only 2.2% error. The third task was the so called 'E-set' speech recognition test, which requires differentiation between the names of the letters in the alphabet which end in e (b,c,d,e,g,p,t,v,z). Fifty male and fifty female speakers spoke the letters in isolated fashion over dial-up telephone lines. The audio was sampled at 6.67 KHz and 24 cepstral features (12 cepstrum + 12 delta-cepstrum) were extracted. A conventional DTW reached 59.8% accuracy, which was improved to 63.9% by a perceptron trained by least squared error, and to 70% when GPD was employed.

The present author has often found that when the MLP has relatively few hidden units the MCE criterion tends to cause the network to get caught in a relatively poor local minimum, while the squared error criterion fairs better on the average. This is due to the aforementioned fact that *very* incorrectly classified patterns are not corrected. This can be easily remedied by initially training with squared error and polishing off with MCE.

4.2 DTW

At first glance it would seem to be impossible for DTW based classifiers to take advantage of the MCE/GPD training techniques, since they are not usually trained at all. DTW classifiers are conventionally constructed by simply selecting one or more templates per word (or whatever phonetic unit is desired to be recognized), or by clustering and selecting cluster centers, with most computation being relegated to the recognition phase. However, where there is a will there is a way, in fact in this case at least two ways [3, 4, 9, 11]. For MCE/GPD training of DTW classifiers, the reference template parameters are gradually adapted in accordance with input patterns. As is always the case, the classification phase is uneffected by the use of MCE.

We will take the DTW parameter vector to be $\mathbf{W} = \{r_{kb}, w_{kb}\}$ where $k = 1 \ldots m$ is the word index, $b = 1 \ldots b_k$ is the utterance of word k index, r_{kb} is the variable-length reference template of feature vectors for utterance kb, and w_{kb} is an (optional) weighting for this template. We desire an algorithm which will adapt $\mathbf{W}$, that is both the template and its weighting, with each presentation of a training pattern.

The basic discriminant function is somewhat more complex than those we have considered before. We need some way of utilizing the DTW distortions (path scores), which, for word k, template b and path θ we denote (following [4, 9])

$$D_\theta(\mathbf{x}; r_{kb}, w_{kb}) = \frac{1}{F_{kb}} \sum_{f=1}^{F_{kb}} w_{kbf} ||x_{\theta f} - r_{kbf}||^2$$

where F_{kb} is the number of frames in template kb, $f = 1 \ldots F_{kb}$ is the running frame index, w_{kbf} is the relative importance weighting for frame f in template kb, $x_{\theta f}$ is the input frame which corresponds to r_{kbf} along DTW path θ, and we can take the frame distance to be squared Euclidean

$$||x - r||^2 = \sum_{\nu=1}^{n} (x_\nu - r_\nu)^2$$

where $\nu = 1 \ldots n$ is the input feature index. In [9] the importance weightings w_{kb} are neglected (taken to be unity) and only the template parameters r_{kb} are trained, conversely in [3] the templates are determined via k-means clustering and *only* the weights are trained.

We *could* consider, for each word, the best path for the best template. However, in keeping with MCE philosophy, in order to ensure continuity and differentiability of the resulting cost functions, it is necessary to form weighted sums of the type one employs in the definition of the misclassification measure. We do this, as usual, in stages.

First one defines the *reference distance* between input $\mathbf{x}$ and template kb as

$$D(\mathbf{x}; r_{kb}, w_{kb}) = \left[\frac{1}{\Theta_{kb}} \sum_{1}^{\Theta_{kb}} \{D_\theta(\mathbf{x}; r_{kb}, w_{kb})\}^{-\xi} \right]^{-\frac{1}{\xi}}$$

where the sum is over all Θ_{kb} possible DTW paths. ξ is some positive number, and the exponents are taken negative due to the fact that smaller distances correspond to better matches. One can of course restrict ones attention to the most important paths for large ξ, and in the limit $\xi \to \infty$ only the optimal path contributes. (Were one to use positive exponents the worst path would dominate!)

Now one defines the discriminant as the *class distance* between input $\mathbf{x}$ and word k

$$g_k(\mathbf{x}; \mathbf{W}) = \left[\frac{1}{b_k} \sum_{1}^{b_k} \{D(\mathbf{x}; r_{kb}, w_{kb})\}^{-\zeta} \right]^{-\frac{1}{\zeta}}$$

where the sum is over all b_k templates belonging to word k. For large ζ only those templates with small DTW distortion contribute, with the closest template dominating the sum for $\zeta \to \infty$.

Next one defines the misclassification measure to be

$$d_k(\mathbf{x}; \mathbf{W}) = -g_k(\mathbf{x}; \mathbf{W}) + \left[\frac{1}{m-1} \sum_{j \neq k} g_j(\mathbf{x}; \mathbf{W})^{-\eta} \right]^{\frac{1}{-\eta}}$$

and the loss and cost functions are defined as usual. One can now carry out the differentiations in equation (22) or (26) in order to obtain explicitly an iterative correction algorithm for the parameters r_{kb} and w_{kb}. The exact expressions are rather unenlightening and will not be quoted here.

The alternative implementation of GPD for DTW [11] exploits the similarity between a particular version of LVQ (Learning Vector Quantization) and GPD. First a shift invariant phoneme detector is trained based on the discriminant

$$g_k(\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_F; \mathbf{W}) = \sum_{f=1}^{F} \left[\sum_{1}^{m} ||x_f - r_k||^{-\zeta} \right]^{-\frac{1}{\zeta}}$$

where F input feature vectors are taken into account, and r_k are phoneme templates. The training is carried out via a generalization of LVQ3 with GPD criterion, and the resulting classifier is called the PBMEC (Prototype Based Minimum Error Classifier). Next the words are considered to be phoneme sequences, with parameters shared for the same phoneme in different words. A time window is shifted over the input and for each word, a matrix of frame positions vs phonemes is constructed containing the input to template distances. DTW is then used to find an optimal path, and during the training phase phoneme templates are corrected along this path (more generally all paths should be considered as above). The resulting architecture is similar to the MSTDNN and will not be further described here.

We conclude this subsection with a few selected experimental results. A telephone-line E-set task on which a conventional speech recognizer performs 67.6% classification was improved by [3] to 78.1%. For a similar problem with base-line performance of 61.7% [9] reports 76.7% classification. [4] similarly considers this same benchmark and obtains a best result of 83.7%.

4.3 HMM

In order to apply MCE/GPD to hidden Markov models, we must first define an appropriate discriminant function. We would like to take the discriminant to be the log-likelihood score of the input utterance for the k'th word model along its optimal path Θ_k [5]

$$g_k(\mathbf{x}; \mathbf{W}) = \log P(\mathbf{x}, \Theta_k | \lambda_k) = \log \pi^k_{\theta_0} + \sum_{f=1}^{F(\mathbf{x})} \log a^k_{\theta^k_{f-1} \theta^k_f} + \sum_{f=1}^{F(\mathbf{x})} \log b^k_{\theta^k_f}$$

where $F(\mathbf{x})$ is the number of frames in the input, θ^k_f is the state sequence of the optimal path Θ_k, and the k^{th} model's parameters λ_k consist of π, a and b which are the initial, transition and output probabilities respectively.

The problem with such a *Viterbi*-like approach is that while changes in the π, a and b parameters are smooth, changes in most likely path are discrete in nature. Thus, with such an approach, one would not be able to differentiate the resulting cost function. The alternative is to use a *Baum-Welch*-like approach, wherein we consider the likelihoods of *all* legitimate paths

$$g_k(\mathbf{x}; \mathbf{W}) = \log \sum_{\Theta_k} P(\mathbf{x}, \Theta_k | \lambda_k) \,.$$

As usual, the system will classify the input as belonging to word k only if the discriminant is greater than that of all other models. We must next define a misclassification measure based on the discriminant, and proceed to define the loss and cost functions as usual. In the misclassification measure the a and b parameters will appear raised to some power η (with of course the corresponding $\frac{1}{\eta}$ outside the sum). When $\eta \to \infty$ the likely (Viterbi) path dominates, and the two approaches above become the same. Iterative gradient descent correction formulas can be derived which replace the usual (Baum-Welch) re-estimation formulas. There are many different types of HMM models, and accordingly many variants of the MCE/GPD approach to them. Detailed description of two such approaches are given in [5, 12]. It will suffice to quote two experimental results. Ten cycles of segmental-GPD training [5] increased a telephone channel E-set recognizer (10-state 5-mixture Left-Right) from 76% to 88.3% recognition, which these authors claim to be the best attained on this set. On the TI connected digit database, base-line 98.7% was improved to 98.8% by this same method.

5 Application to Open Set Problems

Up to now we have dealt with 'closed set' problems, ie. problems wherein the input to the classifier always belongs to one of the classes to be recognized. For such problems the only type of error is misclassification. Many real-world problems are of the 'open-set' variety, ie. inputs may not belong to any of the classes. In such problems we have two types of errors, misclassifications and false alarms. For example, when classifying phonemes in continuous speech or characters in cursive script, we have no general method of a priori faultless segmentation, and may elect to require the classifier itself to perform this function. Such a classifier is usually called a *spotter*. If a spotter identifies a region between two phonemes / characters as valid, a false alarm results. In order to deal with false alarms we must slightly change the four functions of section 3 in order to obtain the Minimum SPotting Error (MSPE) criterion [10].

In order to be specific we will tailor our presentation to speech spotting problems (which can be word spotting or phoneme spotting). First we will define our nomenclature and notation. The input (eg. incoming speech) is divided into *frames*, and S-dimensional feature vectors are extracted from each frame. The feature vector of frame i will be denoted $x_i = (x_{i_1}, x_{i_2} \dots x_{i_S}) \epsilon \Re^S$. A *block* b is the sequence of feature vectors between frames p_b and q_b which is of length $I_b = q_b - p_b + 1$, namely $\mathbf{x} = \{x_i\}_{i=p_b}^{q_b}$. We will assume a somewhat specific architecture for the spotter, that of one model per class (eg. word or phoneme) to be spotted, each of which has its own decision threshold. The parameter vector of the entire spotter thus divides up into m sub-spotters $\mathbf{W} = \{w^k, h^k\}_{k=1}^m$ where w^k are the parameters of the class model and h^k is the decision threshold.

The discriminant function, $g_k^i(\mathbf{x}, w^k)$, indicates the similarity in frame i between the input $\mathbf{x}$ and the k'th class. The *spotting measure* $d_k^b(\mathbf{x}, \mathbf{W})$ is positive when the class k is

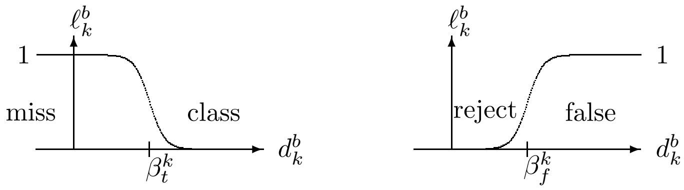

Figure 3: The MSPE loss function for actual examples (left) and false alarms (right)

deamed to be present in block b, and is defined by

$$d_k^b(\mathbf{x}, \mathbf{W}) = h^k - \ln \left\{ \frac{1}{I_b} \sum_{i=p_b}^{q_b} e^{[g_k^i(\mathbf{x}, w^k)]^{-\xi}} \right\}^{-\frac{1}{\xi}} . \tag{40}$$

The calculation of this measure entails computation of the discriminants corresponding to class k for all frames in block b (from p_b to q_b), plugging them into a MCE-like averaging formula (averaging over frames in a block) and comparing the logarithm of this formula with the decision threshold. Note that the spotting measure has an interpretation which is somewhat different from that of the misclassification measure in two respects. First, higher spotting measures imply more likely word appearances (ie. the spotting measure is actually akin to a *classification* measure). Secondly, rather than comparing one class to the others, we compare the class to a class-dependent threshold. This results from the assumed sub-spotter architecture and allows consistent treatment of false alarms and misclassifications, at the expense of adding m new parameters to be determined.

The loss function must now take both misclassifications and false alarms into account, and so is defined as

$$\ell_k^b(\mathbf{x}, \mathbf{W}) = \begin{cases} \gamma_t^k \sigma_{\alpha_t^k \beta_t^k}(d_k^b(\mathbf{x}, \mathbf{W})) & \text{if class } k \text{ really in } b \\ \gamma_f^k \sigma_{\alpha_f^k \beta_f^k}(d_k^b(\mathbf{x}, \mathbf{W})) & \text{else} \end{cases} . \tag{41}$$

Here the sigmoid is given by

$$\sigma_{\alpha\beta}(z) = \frac{1}{1 + e^{\alpha(z-\beta)}}$$

the cutoffs β_t^k and β_f^k relate to the discriminant values which should differentiate between detections and rejections, the weights $\gamma_t^k > 0$ and $\gamma_f^k > 0$ determine the relative importance of a misclassification to a false alarm, while the inverse temperatures $\alpha_t^k > 0$ and $\alpha_f^k < 0$ regulate the smoothing of the step functions (the signs are chosen in such a fashion that missing an actual example and detecting a false alarm cause positive loss). The loss function for the two cases is depicted in figure 3.

The total cost is now defined as the sum over all blocks for all words

$$L(\mathbf{W}) = \sum_{k=1}^{m} \sum_{b=1}^{B} \ell_k^b(\mathbf{x}, \mathbf{W}) \tag{42}$$

which, for appropriate γ_t^k and γ_f^k, will approximate the number of errors (both misses and false alarms, weighted accordingly). We can now update the parameters $\mathbf{W}$ by MCE or GPD, which simultaneously updates both the class model parameters and the decision thresholds.

Komori and Katagiri [10] conducted a series of phoneme spotting experiments in which 18 English language consonants were spotted in high quality speech. 16 mel-scale spectral coefficients were input every 5 msec. and the templates were initialized by k-means clustering. The spotters were all of the same size and 5 templates were used per phoneme. The following table, adapted from [10] shows the effect of 160 epochs of MSPE training on the test set.

phoneme	**misdetection(%)**		false alarms/hour/phoneme	
	before MSPE	after MSPE	before MSPE	after MSPE
p	60.9	47.8	12.0	0.7
t	53.9	2.0	0.3	1.2
k	24.1	12.1	11.9	2.1
b	94.4	61.1	242.2	1.7
d	100.0	22.9	88.6	2.9
g	64.3	85.7	13.7	0.0
m/n	9.9	4.7	28.6	9.5
N	34.0	35.9	22.9	10.8
s	10.1	13.0	3.5	2.4
sh	100.0	4.1	5.3	1.2
h	40.9	30.3	18.8	2.3
z	7.4	29.6	13.6	0.0
dj	18.6	20.9	5.3	1.4
w	25.0	25.0	25.1	1.3
r	58.1	21.4	17.4	2.6
j	56.3	28.7	11.6	3.7
ts	100.0	38.5	0.0	0.0
ch	14.8	25.9	21.0	1.1

Table Results of Komori and Katagiri for phoneme spotting.

6 Conclusions

The basic philosophy behind the MCE approach is that a classifier should be trained using a criterion which is as similar as possible to that actually used in verifying its worth – usually the percentage of errors it makes. Thus MCE is more theoretically consistent than ad hoc training criteria such as minimum squared error. However, in addition to this theoretical advantage, there are several practical reasons for considering MCE training.

- The MCE/GPD approach provides a systematic procedure for deriving smooth cost functions. Thus any classifier for which a MCE-compatible discriminant can be defined can be trained using gradient descent techniques.
- Results of proper employment of MCE/GPD techniques to speech recognition problems have been shown to have significantly better performance (ten to fifteen percent in typical cases).
- Retrofitting of MCE to existing systems usually requires only minor adjustments. The actual running of the classifier (which may have been optimally implemented) is unaffected, only certain aspects of the off-line training are modified.
- Use of MCE on open set problems provides a method of training which utilizes all the training set input rather than just classifiable examples. This leads to superior false alarm rejection. In addition, optimal class dependent thresholds of spotters can be determined.

References

[1] S. Amari (1967), *A Theory of Adaptive Pattern Classifiers*, *IEEE Trans. Elect. Comp.* **EC-16** 299-307.

[2] Alain Biem & Shigeru Katagiri (1993), *Feature Extraction Based on Minimum Classification Error / Generalized Probabilistic Descent Method*, *ICASSP-93* II-275 - II-278.

[3] Pao-Chung Chang, Sin-Horng Chen & Biing-Hwang Juang (1991), *Discriminative Analysis of Distortion Sequences in Speech Recognition*, *ICASSP-91* 549-552.

[4] Pao-Chung Chang & Biing-Hwang Juang (1993), *Discriminative Training of Dynamic Programming Based Speech Recognizers*, *IEEE Trans. Speech & Audio Processing* **1**, 135-143.

[5] W.Chou, B.H. Juang & C.H. Lee (1992), *Segmental GPD Training of HMM Based Speech Recognizer*, *ICASSP-92* I-473 – I-476.

[6] Herbert Gish (1990), *A Probabilistic Approach to the Understanding and Training of Neural Network Classifiers*, *ICASSP-90* 1361-1364.

[7] Biing-Hwang Juang & Shigeru Katagiri (1991), *A Minimum Error Approach to Speech and Pattern Recognition*, *Asilomar-91*, 959-963.

[8] Biing-Hwang Juang & Shigeru Katagiri (1992), *Discriminitive Learning for Minimum Error Classification*, IEEE Trans. *Signal Processing* **40**, 3043-3054.

[9] Takashi Komori & Shigeru Katagiri (1992), *Application of a Generalized Probabilistic Descent Method to Dynamic Time Warping Based Speech Recognition*, *ICASSP-92* I-497 - I-500.

[10] Takashi Komori & Shigeru Katagiri (1993), *An Optimal Learning Method for Minimizing Spotting Errors*, *ICASSP-93* II-271 - II-274.

[11] Erik McDermott & Shigeru Katagiri (1993), *Prototype-Based Discriminative Training for Various Speech Units*, *ICASSP-93* II-271 - II-274.

[12] Neri Merhav (1994), *MCE Training for HMMs*, *preprint*.

[13] Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley and Bruce W. Suter (1990), *The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function*, *IEEE Trans. Neural Networks* **1** 296-298.

[14] Sara A. Solla, Esther Levin and Michael Fleisher (1988), *Accelerated Learning in Layered Neural Networks*, *Complex Systems* **2** 625-640.

[15] M. Sugiyama & K. Kurinami (1993), *Minimal Classification Error Optimization for a Speaker Mapping Neural Network* Proc. 1992 *IEEE-SP* Workshop 233 - 242.