

## Discriminant Pattern Recognition Using Transformation-Invariant Neurons

Diego Sona

Alessandro Sperduti

Antonina Starita

*Dipartimento di Informatica, Università di Pisa, 56125, Pisa, Italy*

To overcome the problem of invariant pattern recognition, Simard, LeCun, and Denker (1993) proposed a successful nearest-neighbor approach based on tangent distance, attaining state-of-the-art accuracy. Since this approach needs great computational and memory effort, Hastie, Simard, and Säckinger (1995) proposed an algorithm (HSS) based on singular value decomposition (SVD), for the generation of nondiscriminant tangent models. In this article we propose a different approach, based on a gradient-descent constructive algorithm, called TD-Neuron, that develops discriminant models. We present as well comparative results of our constructive algorithm versus HSS and learning vector quantization (LVQ) algorithms. Specifically, we tested the HSS algorithm using both the original version based on the two-sided tangent distance and a new version based on the one-sided tangent distance. Empirical results over the NIST-3 database show that the TD-Neuron is superior to both SVD- and LVQ-based algorithms, since it reaches a better trade-off between error and rejection.

### 1 Introduction ---

In several pattern recognition systems the principal and most desired feature is robustness against transformations of patterns. Simard, LeCun, and Denker (1993) partially solved this problem by proposing the tangent distance as a classification function invariant to small transformations. They used the concept in a nearest-neighbor algorithm, achieving state-of-the-art accuracy on isolated handwritten character recognition. However, this approach has a quite high computational complexity due to the large number of Euclidean and tangent distances that need to be calculated.

Different researchers have shown how such complexity can be reduced at the cost of increased space complexity. Simard (1994) proposed a filtering method based on multiresolution and a hierarchy of distances, while Sperduti and Stork (1995) devised a graph-based method for rapid and accurate search through prototypes.

Different approaches to the problem, aiming at the reduction of classification time and space requirements, while trying to preserve the same

accuracy, have been studied. Specifically, Hastie, Simard, and Säckinger (1995) developed rich models for representing large subsets of the prototypes through a singular value decomposition (SVD)-based algorithm, while Schwenk and Milgram (1995b) proposed a modular classification system (Diabolo) based on several autoassociative multilayer perceptrons, which use tangent distance as the error reconstruction measure. A different but related approach has been pursued by Hinton, Dayan, and Revow (1997), who propose two different methods for modeling the manifolds of data. Both methods are based on locally linear low-dimensional approximations to the underlying data manifolds.

All the above models are nondiscriminant.<sup>1</sup> Although nondiscriminant models have some advantages over discriminant models, (Hinton et al., 1997), the amount of computation during recognition is usually higher for nondiscriminant models, especially if a good trade-off between error and rejection is required. On the other hand, discriminant models take more time to be trained. In several applications, however, it is more convenient to spend extra time for training, which is usually performed only once or a few times, so as to have a faster recognition process, which is repeated millions of times. In this cases, discriminant models should be preferred.

In this article, we discuss a constructive algorithm for the generation of discriminant tangent models.<sup>2</sup> The proposed algorithm, which is an improved version of the algorithm previously presented in Sona, Sperduti, and Starita (1997), is based on the definition of the TD-Neuron (TD stands for "tangent distance") where the net input is computed by using the one-sided tangent distance instead of the standard dot product. Using this definition, we have devised a constructive algorithm, which we compare here with HSS and learning vector quantization (LVQ) algorithms. In particular, we report results obtained for the HSS algorithm using both the original version based on the two-sided tangent distance and a new version based on the one-sided tangent distance. For comparison, we also present the results of the LVQ2.1 algorithm, which turned out to be the best among the LVQ algorithms.

The one-sided version of the HSS algorithm was derived in order to have a fair comparison against TD-Neuron, which exploits the one-sided tangent distance. Empirical results over the NIST-3 database of handwritten digits show that the TD-Neuron is superior to both HSS algorithms and LVQ algorithms since it reaches a better trade-off between error and rejection. More surprising, our results show that the one-sided version of the HSS algorithm is superior to the two-sided version, which performs poorly when introducing a rejection class. An additional advantage of the proposed algorithm is the constructive approach.

---

<sup>1</sup> Schwenk and Milgram (1995a) proposed a discriminant version of Diabolo as well.

<sup>2</sup> In the sense that the model for each class is generated taking into account negative examples also, that is, examples of patterns belonging to the other classes.

In Sections 2 and 3, we give an overview of tangent distance and tangent distance models, respectively, and define a novel version of the HSS algorithm, based on one-sided tangent distance. A new formulation for discriminant tangent distance models is proposed in section 4, while the proposed TD-Neuron model is presented in section 5, which also includes details on the training algorithm. Comparative empirical results on a handwritten digit recognition task between our algorithm, HSS algorithms, LVQ, and nearest-neighbor algorithms with Euclidean distance are presented in section 6. Finally, a discussion of the results and conclusions are reported in section 7.

## 2 Tangent Distance Overview

---

Consider a pattern recognition problem where invariance for a set of  $n$  different transformations is required. Given an image  $\mathbf{X}_i$ , the function  $\mathbf{X}_i(\boldsymbol{\theta})$  is a manifold of at most  $n$  dimensions, representing the set of patterns that can be obtained by transforming the original image through the chosen transformations, where  $\boldsymbol{\theta}$  is the amount of transformations and  $\mathbf{X}_i = \mathbf{X}_i(\mathbf{0})$ . The ideal would be to use the transformation-invariant distance,

$$D_I(\mathbf{X}_i, \mathbf{X}_j) = \min_{\boldsymbol{\alpha}, \boldsymbol{\theta}} \|\mathbf{X}_i(\boldsymbol{\alpha}) - \mathbf{X}_j(\boldsymbol{\theta})\|.$$

However, the formalization of the manifold equation and, in particular, the computation of the distance between the two manifolds, is very hard. For this reason, Simard et al. (1993) proposed an approach based on the local linear approximation of the manifold by

$$\tilde{\mathbf{X}}_i(\boldsymbol{\theta}) = \mathbf{X}_i + \sum_{j=1}^n \mathbf{T}_{\mathbf{X}_i}^j \theta_j,$$

where  $\mathbf{T}_{\mathbf{X}_i}^j$  are  $n$  different tangent vectors at the point  $\mathbf{X}_i(\mathbf{0})$ , which can easily be computed by finite difference. The distance between the two manifolds is then approximated by the so-called tangent distance (Simard et al., 1993):

$$D_T(\mathbf{X}_i, \mathbf{X}_j) = \min_{\boldsymbol{\alpha}, \boldsymbol{\theta}} \|\tilde{\mathbf{X}}_i(\boldsymbol{\alpha}) - \tilde{\mathbf{X}}_j(\boldsymbol{\theta})\|. \quad (2.1)$$

Of course, the approximation is accurate only for local transformations; however, in character recognition problems, global invariance may not be desired, since it can cause confusion between patterns such as  $n$  and  $u$ .

The tangent distance defined by equation 2.1 is called two-sided tangent distance, since it is computed between two subspaces. There exists also a less computationally expensive version, one-sided tangent distance (Schwenk

& Milgram, 1995a), where the distance is computed between a subspace and a pattern in the following way:

$$D_T^{1\text{-sided}}(\mathbf{X}_i, \mathbf{X}_j) = \min_{\alpha} \|\tilde{\mathbf{X}}_i(\alpha) - \mathbf{X}_j\|. \tag{2.2}$$

### 3 Tangent Distance Models

---

The main drawback of tangent distance is its high computational requirement, if compared with Euclidean distance. For this reason, several authors have tried to devise compact models, based on tangent distance, that can summarize relevant information conveyed by a set of patterns.

To address this problem, Hastie et al. (1995) proposed an algorithm for the generation of rich models representing large subsets of patterns. Given a set of patterns  $\{\mathbf{X}_1, \dots, \mathbf{X}_{N_C}\}$  of class C, they proposed the tangent subspace model,

$$\mathbf{M}(\boldsymbol{\theta}) = \mathbf{W} + \sum_{i=1}^n \mathbf{T}_i \theta_i,$$

where  $\mathbf{W}$  is the centroid and the set  $\{\mathbf{T}_i\}$  constitutes the associated invariant subspace of dimension  $n$ .

According to this definition, for each class C, the model  $\mathbf{M}_C$  can be computed as

$$\mathbf{M}_C = \arg \min_{\mathbf{M}} \sum_{p=1}^{N_C} \min_{\boldsymbol{\theta}_p, \alpha_p} \|\mathbf{M}(\boldsymbol{\theta}_p) - \mathbf{X}_p(\alpha_p)\|^2, \tag{3.1}$$

minimizing the error function over  $\mathbf{W}$  and  $\mathbf{T}_i$ .

The above definition constitutes a difficult optimization problem; however, it can be solved for a fixed value of  $n$  (i.e., the subspace dimension) by an iterative algorithm based on singular value decomposition, as proposed by Hastie et al. (1995).

If the problem is formulated using the one-sided tangent distance, then equation 3.1 becomes

$$\mathbf{M}_C = \arg \min_{\mathbf{M}} \sum_{p=1}^{N_C} \min_{\boldsymbol{\theta}_p} \|\mathbf{M}(\boldsymbol{\theta}_p) - \mathbf{X}_p\|^2, \tag{3.2}$$

which can be easily solved by principal component analysis theory, also called Karhunen-Loève expansion. In fact, equation 3.2 can be minimized by choosing  $\mathbf{W}$  as the average over all available samples  $\mathbf{X}_p$  and  $\mathbf{T}_i$  as the most representative eigenvectors (principal components) of the covariance matrix  $\Sigma$ , where

$$\Sigma = \frac{1}{N_C} \sum_{p=1}^{N_C} (\mathbf{X}_p - \mathbf{W})(\mathbf{X}_p - \mathbf{W})^T.$$

We will refer to the two versions of the algorithms as HSS, and when necessary we will specify which one is used (one-sided or two-sided).

By construction, the HSS algorithms return nondiscriminant models. In fact, they use only the evidence provided by positive examples of the target class. Moreover, the two-sided HSS algorithm can be used only if a priori knowledge on invariant transformations is present. If this knowledge is not present, the introduction of invariance with respect to an arbitrary transformation can be risky, since this can remove information relevant for the classification task. In this situation, it is preferable to use the one-sided version, which does not commit to any specific transformation.

#### 4 A General Formulation

---

There are good reasons for using discriminant models. Although Schwenk and Milgram (1995a) suggested how to modify the learning rule of Diablo to obtain discriminant models, they never proposed a formalization of discriminant models using tangent distance. In this section, we present a general formulation that allows the user to develop discriminant or nondiscriminant tangent models.

To be able to devise discriminant models, equation 3.1 must be modified in such a way to take into account that all available data must be used during the generation process. The basic idea is to define a model for class  $C$  that minimizes the tangent distances from patterns belonging to  $C$  and maximizes the tangent distances from patterns not in  $C$  (i.e., in  $\bar{C}$ ). Mathematically this can be expressed, for each class  $C$ , by

$$M_C = \arg \min_M \left[ \sum_{p=1}^{N_C} D_T(M, \mathbf{X}_p^C) - \lambda \sum_{p=1}^{N_{\bar{C}}} D_T(M, \mathbf{X}_p^{\bar{C}}) \right], \quad (4.1)$$

where  $M$  is the generic model  $\{\mathbf{W}, \mathbf{T}_1, \dots, \mathbf{T}_n\}$ ,  $N_C$  is the number of patterns  $\mathbf{X}_p^C$  belonging to the class  $C$ , and  $N_{\bar{C}}$  is the number of patterns  $\mathbf{X}_p^{\bar{C}}$  not belonging to  $C$ .

Note that the second sum is multiplied by a constant  $\lambda$ , which identifies how much discriminant the model should be. If  $\lambda = 0$ , equation 4.1 becomes equal to equation 3.1 (or 3.2 when considering the one-sided tangent distance). On the other hand, if  $\lambda$  is large, the resulting model may not be a good descriptive model for class  $C$ . In any case, no bounded solution to equation 4.1 may exist if the term associated with  $\lambda$  is not bounded.

#### 5 TD-Neuron

---

The TD-Neuron is so called because it can be considered as a neural computational unit that computes (as net input) the square of the one-sided tangent distance of the input vector  $\mathbf{X}_k$  from a prototype model defined by a set of

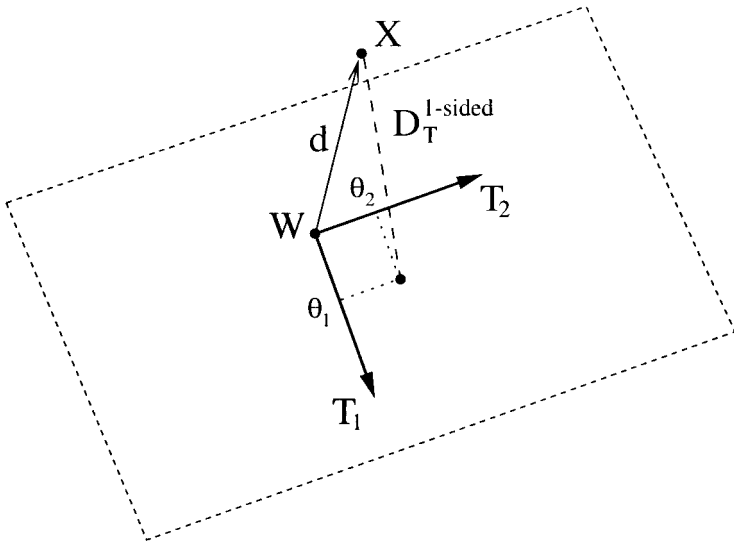


Figure 1: Geometric interpretation of equation 5.2. Note that  $W$  and  $T_i$  span the invariance manifold,  $d$  is the Euclidean distance between the pattern  $X$  and the centroid  $W$ , and  $net = (D_T^{1\text{-sided}})^2$  is the one-sided tangent distance.

internal parameters (weights). Specifically, it is characterized by a set of  $n + 1$  vectors of the same dimension as the input vectors. One vector ( $W$ ) is used as the reference vector (centroid), while the remaining vectors  $\{T_1, \dots, T_n\}$  are used as tangent vectors. Moreover, the set of tangent vectors constitutes an orthonormal basis.

This set of parameters is organized in such a way as to form a tangent model. Formally, the net input of a TD-Neuron for a pattern  $k$  is

$$net_k = \min_{\theta} \|M(\theta) - X_k\|^2 + \beta, \tag{5.1}$$

where  $\beta$  is the offset. A good model should return small net input for patterns belonging to the learned class.

Since the tangent vectors constitute an orthonormal basis, equation 5.1 can exactly and easily be computed by using the projections of the input vector over the model subspace (see Figure 1):

$$net_k = \underbrace{\|X_k - W\|}_{d_k}^2 - \sum_{i=1}^n [(X_k - W)^t T_i]^2 + \beta$$

$$= \mathbf{d}_k^t \mathbf{d}_k - \sum_{i=1}^n [\underbrace{\mathbf{d}_k^t \mathbf{T}_i}_{\gamma_{ik}}]^2 + \beta, \quad (5.2)$$

where, for the sake of notation,  $\mathbf{d}_k$  denotes the difference between the input pattern  $\mathbf{X}_k$  and the centroid  $\mathbf{W}$ , and the projection of  $\mathbf{d}_k$  over the  $i$ th tangent vector is denoted by  $\gamma_{ik}$ .

Note that the right side of equation 5.2 mainly involves dot products, just as in a standard neuron.

The output of the TD-Neuron is then computed by transforming the net through a nonlinear monotone function  $f$ . In our experiments, we have used the symmetric sigmoidal function

$$o_k = \frac{2}{1 + e^{net_k}} - 1. \quad (5.3)$$

We have used a monotonic decreasing function so that the output corresponding to patterns belonging to the target class will be close to 1.

**5.1 Training the TD-Neuron.** A discriminant model based on the TD-Neuron can be obtained by adapting equation 4.1. Given a training set  $\{(\mathbf{X}_1, t_1), \dots, (\mathbf{X}_N, t_N)\}$ , where

$$t_i = \begin{cases} 1 & \text{if } \mathbf{X}_i \in C \\ -1 & \text{if } \mathbf{X}_i \in \bar{C} \end{cases}$$

is the  $i$ th desired output for the TD-Neuron, and  $N = N_C + N_{\bar{C}}$  is the total number of patterns in the training set, an error function can be defined as

$$\mathcal{E} = \frac{1}{2} \sum_{k=1}^N (t_k - o_k)^2, \quad (5.4)$$

where  $o_k$  is the output of the TD-Neuron for the  $k$ th input pattern.

Equation 5.4 can be written into a form similar to equation 4.1 by making explicit the target values  $t_i$ , by splitting the sum in such a way to group together patterns belonging to  $C$  and patterns belonging to  $\bar{C}$ , and by weighting the negative examples with a constant  $\lambda \geq 0$ :

$$\mathcal{E} = \frac{1}{2} \left[ \sum_{k=1}^{N_C} (1 - o_k)^2 + \lambda \sum_{j=1}^{N_{\bar{C}}} (1 + o_j)^2 \right]. \quad (5.5)$$

In our experiments we have chosen  $\lambda = \frac{N_C}{N_{\bar{C}}}$ , thus balancing the strength of patterns belonging to  $C$  and those belonging to  $\bar{C}$ .

Using equations 5.2 and 5.3, it is trivial to compute the changes for the centroid, the tangent vectors, and the offset by using a gradient-descent approach over equation 5.5:

$$\Delta \mathbf{W} = -\eta \left( \frac{\partial \mathcal{E}}{\partial \mathbf{W}} \right) = -2 \eta \sum_{k=1}^N \left[ (t_k - o_k) f'_k \left( \mathbf{d}_k - \sum_{i=1}^n \gamma_{ik} \mathbf{T}_i \right) \right] \quad (5.6)$$

$$\Delta \mathbf{T}_i = -\eta \left( \frac{\partial \mathcal{E}}{\partial \mathbf{T}_i} \right) = -2 \eta \sum_{k=1}^N [(t_k - o_k) f'_k \gamma_{ik} \mathbf{d}_k] \quad (5.7)$$

$$\Delta \beta = -\eta_\beta \left( \frac{\partial \mathcal{E}}{\partial \beta} \right) = \eta_\beta \sum_{k=1}^N [(t_k - o_k) f'_k] \quad (5.8)$$

where  $\eta$  and  $\eta_\beta$  are learning parameters, and  $f'_k = \frac{\partial o_k}{\partial net_k}$ .

Before training the TD-Neuron by gradient descent, however, the tangent subspace dimension must be decided. To solve this problem, we developed a constructive algorithm that adds tangent vectors one by one, according to computational needs. This idea is also justified by the observation that using equations 5.6 through 5.8 leads to the sequential convergence of the tangent vectors according to their relative importance.

This means that in the first approximation, all the tangent vectors remain random vectors while the centroid converges first. Then one of the tangent vectors converges to the most relevant transformation (while the remaining tangent vectors are still immature), and so on until all the tangent vectors converge, one by one, to less and less relevant transformations.

This behavior suggests starting the training using only the centroid (i.e., without tangent vectors) and then adding tangent vectors as needed. Under this learning scheme, since there are no tangents when the centroid is computed, equation 5.6 becomes

$$\Delta \mathbf{W} = -\eta \left( \frac{\partial \mathcal{E}}{\partial \mathbf{W}} \right) = -2\eta \sum_{k=1}^N [(t_k - o_k) f'_k \mathbf{d}_k]. \quad (5.9)$$

The constructive algorithm has two phases (see Table 1). First is the centroid computation, based on the iterative use of equations 5.8 and 5.9. Then the centroid is frozen, and one by one all tangent vectors  $\mathbf{T}_i$  are trained using equations 5.7 and 5.8. At each iteration in the learning phase, the tangent vector  $\mathbf{T}_i$  must be orthonormalized with respect to the already computed tangent vectors.<sup>3</sup> If after a fixed number of iterations (we have used 300 iterations) the total error variation is less than a fixed threshold (0.01%) and no change in the classification performance over training set occurs, a new tangent vector is added. The tangent vectors are iteratively added until changes in the classification accuracy become irrelevant.

---

<sup>3</sup> The computational complexity of the orthonormalization is linear in the number of already computed tangent vectors.



Table 1: Constructive Algorithm for the TD-Neuron.

---

Initialize the centroid  $W$ .

Update  $\beta$  and  $W$  by equations 5.8 and 5.9 until they converge.

Freeze  $W$ .

**REPEAT.**

    Initialize a new tangent vector  $T_i$ .

    Update  $T_i$  and  $\beta$  with equations 5.7 and 5.8, and orthonormalize  $T_i$  with respect to  $\{T_1, \dots, T_{i-1}\}$  until it converges.

    Freeze  $T_i$ .

**UNTIL** new  $T_i$  gives little accuracy changes.

---

The initialization of internal vectors of the TD-Neuron can be done in many different ways. On the basis of empirical evidence, we have concluded that the learning phase of the centroid can be considerably reduced by initializing the centroid with the mean value of the patterns belonging to the positive class. We have also devised a “good” initialization algorithm for tangent vectors (see Table 2) that tries to minimize the drop in the net input for all the patterns due to the increase in the tangent subspace dimension (see Figure 2). This is obtained by introducing a new tangent vector that mainly spans the residual subspace between the patterns in the positive class and the current model. In this way, patterns that are in the negative class will be only mildly affected by the new introduced tangent vector.

We have also devised a “better” initialization algorithm based on principal components of difference vectors for all classes. However, the observed training speed-up is not justified by the additional computational overhead due to SVD computation needed at each tangent vector insertion.

In our experiments, we have also used a simple form of regularization over the parameters (weight decay with penalty equal to 0.997, i.e., all pa-

Table 2: Initialization Procedure for the Tangent Vectors.

---

- For each class  $c \in (C \cup \overline{C})$ , compute the mean value of differences between patterns and model:  $d_c = \frac{1}{N_c} \sum_{p=1}^{N_c} (X_p - M(\theta))$ .
- Orthonormalize the vector  $d_c$  of the class  $C$  with respect to the mean values of differences of all other classes belonging to  $\overline{C}$ , and return it as the new initial tangent vector.

---

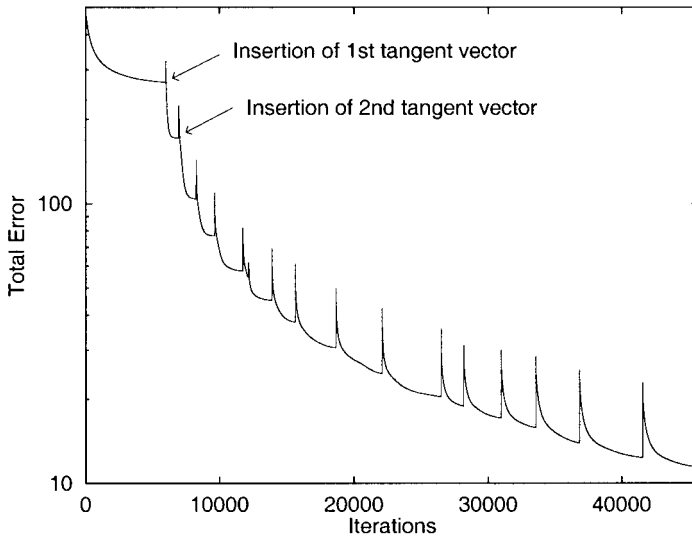


Figure 2: Total error variation during learning phase for pattern 0. At each new tangent insertion, there is a reduction of the distance between patterns and the model (also for patterns belonging to class  $\bar{C}$ ). This affects the output of the neuron for all patterns, increasing the total output squared error.

rameters are multiplied by the penalty before adding the gradient), obtaining a better convergence.

## 6 Results

---

In order to obtain comparable results, we had to use the same number of parameters for all algorithms. In particular, for tangent-based algorithms, the number of vectors is given by the number of tangent vectors incremented by 1 (the centroid). For this reason, the LVQ algorithms are compared to tangent distance-based algorithms using a number of reference vectors equal to the number of tangent vectors plus 1. Furthermore, with the two-sided HSS algorithm, we also have to consider the tangent vectors corresponding to the input patterns. Specifically, we used six transformations (tangent vectors) for each input pattern: clockwise and counterclockwise rotations and translation in the four cardinal directions.<sup>4</sup>

We tested our constructive algorithm versus the two versions of the HSS algorithm and LVQ algorithms in the LVQ\_PAK package (Kohonen, Hynni-

---

<sup>4</sup> We preferred to approximate the exact tangents by finite differences; in this way, the local shape of the manifold is interpolated in a better way.

nen, Kangas, Laaksonen, & Torkkola, 1996) (optimized-learning-rate LVQ1, original LVQ1, LVQ2.1, LVQ3), using 10,704 binary digits taken from the NIST-3 data set. The binary  $128 \times 128$  digits were transformed into 64-gray level  $16 \times 16$  images by a simple local counting procedure.<sup>5</sup> The only preprocessing transformation performed was the elimination of empty borders.

The training set consisted of 5000 randomly chosen digits; the remaining digits were used in the test set. For each tangent distance algorithm, a single tangent model for each class of digit was computed. With the LVQ algorithms, a set of reference vectors was used for each class; the number of reference vectors was chosen so as to have as many parameters as in the tangent distance algorithms. In particular, we tested all algorithms based on tangent distance using a different number of vectors for each experiment, starting from 1 vector per class (centroid without tangent vectors) up to 16 vectors per class (centroid plus 15 tangent vectors). The number of reference vectors for LVQ algorithms has been chosen accordingly.

Concerning LVQ algorithms, here we report just the results obtained by using LVQ2.1 with 1-NN based on Euclidean distance as the classification rule, since this algorithm reached the best performance over an extended set of experiments involving LVQ algorithms with different settings for the learning parameters.

The classification of the test digits was performed using the label of the closest model for HSS, the 1-NN rule for LVQ algorithms, and the highest output for the TD-Neuron algorithm. For the sake of comparison, we also performed a classification using the nearest-neighbor rule (1-NN) with the Euclidean distance as classification metric. In this case, we classified each pattern by looking at the label of the nearest vector in the learning set.

In Figure 3 we report the results obtained on the test set for different numbers of tangent vectors for all models. The best classification result (96.84%) was given by 1-NN with Euclidean distance, followed by the two-sided HSS with 9 tangent vectors (96.6%).<sup>6</sup> With the same number of parameters (15 tangent vectors), the TD-Neuron and the one-sided HSS gave a performance rate of 96.51% and 96.42%, respectively. Finally, LVQ2.1 obtained a performance rate of 96.48% using 15 vectors per class.

From these results, it can be noted that the two-sided HSS algorithm does overfit the data after the ninth tangent vector; this is not true for the remaining algorithms. Nevertheless, all the models reach a similar performance with the same number of parameters, which is slightly below the performance attained by the 1-NN classifier using Euclidean distance. However, both the tangent models and LVQ algorithms have the advantage of being less demanding in both space and response time than 1-NN.

---

<sup>5</sup> The original image is partitioned into  $16 \times 16$  windows and the number of pixel with value equal to 1 is used as the gray value for the corresponding pixel in the new image.

<sup>6</sup> There are also six tangent vectors corresponding to the input pattern.

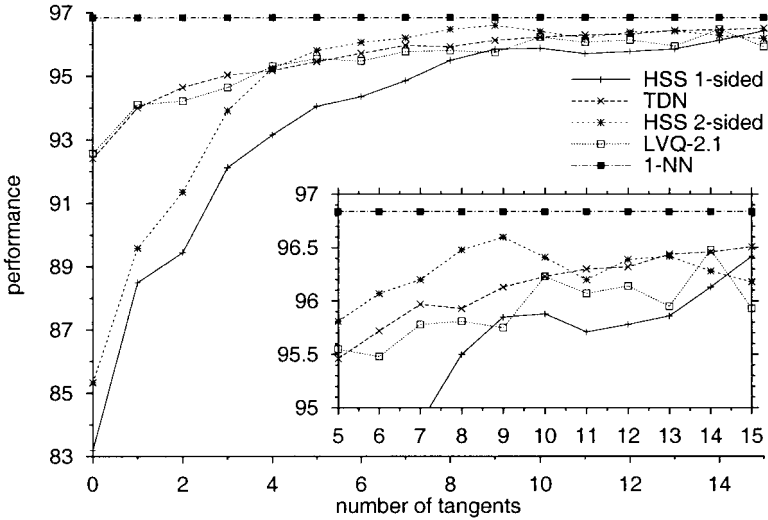


Figure 3: Results obtained on the test set by models generated by both versions of HSS, LVQ2.1, TD-Neuron, and 1-NN with Euclidean distance.

It is interesting that the recognition performance of the TD-Neuron system is basically monotone in the number of parameters (tangent vectors), while all other methods present overfitting and high variance in the results.

Although from Figure 3 it seems that the tangent models and LVQ2.1 are equivalent, when introducing a rejection criterion the model generated by the TD-Neuron outperforms the other algorithms (see Figure 4). We used the same rejection criterion for all algorithms. A pattern is rejected when the difference between the first- and the second-best outputs belonging to different classes is less than a fixed threshold.<sup>7</sup> Furthermore, introducing the rejection criterion also leads to the surprising result that the one-sided version of HSS performs better than the two-sided version.

In order to assess whether the better performance exhibited by the TD-Neuron was due to the specific rejection criterion or the discriminant capability of the model, we performed some experiments using a different rejection criterion: reject when the best output value is smaller (TD-Neuron) or greater (HSS, LVQ and 1-NN) than a threshold. In Figure 5 we report the curves obtained for the best models—TD-Neuron and one-sided HSS—with, for comparison, the corresponding curves of Figure 4.

These results demonstrate that the improvement shown by the TD-Neuron is not tight to a specific rejection criterion. Moreover, removing the

<sup>7</sup> Obviously the threshold is different for each algorithm.

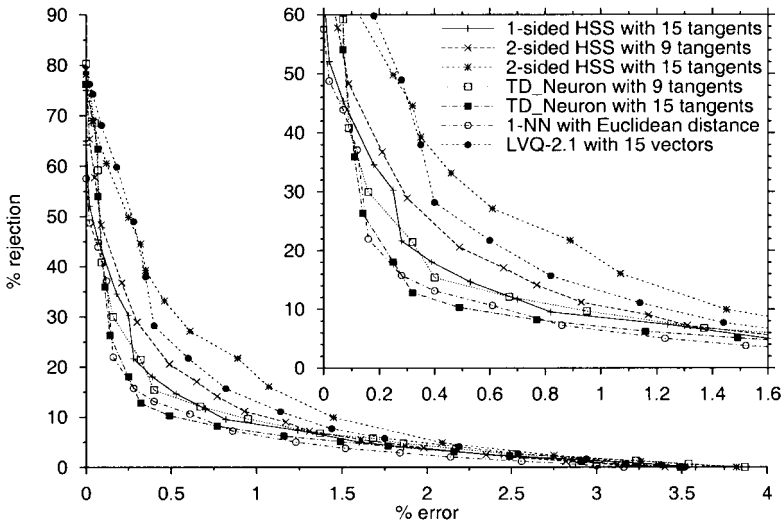


Figure 4: Error-rejection curves for all algorithms: one-sided HSS with 15 tangent vectors, two-sided HSS with 9 and 15 tangent vectors, TD-Neuron with 9 and 15 tangent vectors, LVQ2.1 with 15 reference vectors, and 1-NN with Euclidean distance. The boxed diagram shows a detail of the curves demonstrating that the TD-Neuron with 15 tangent vectors has the best trade-off between error and rejection.

sigmoidal function from the TD-Neuron during the test phase, the rejection curves does not change significantly for both rejection criteria. Thus, we can conclude that the improvement of the TD-Neuron is mainly due to the discriminant training procedure.

In Figure 6, four examples of TD-Neuron models are reported. In the left-most column, the centroids of patterns 0, 1, 2, and 3 are shown. The remaining columns contain the first (and most important) four tangent vectors for each model.

## 7 Discussion and Conclusion

We introduced the TD-Neuron, which implements the one-sided version of the tangent distance, and gave a constructive learning algorithm for building a tangent subspace with discriminant capabilities.

There are many advantages to using the proposed computational model versus the HSS model and LVQ algorithms. Specifically, we believe that the proposed approach is particularly useful in applications where it is very important to have a classification system that is both discriminant and fast in recognition.

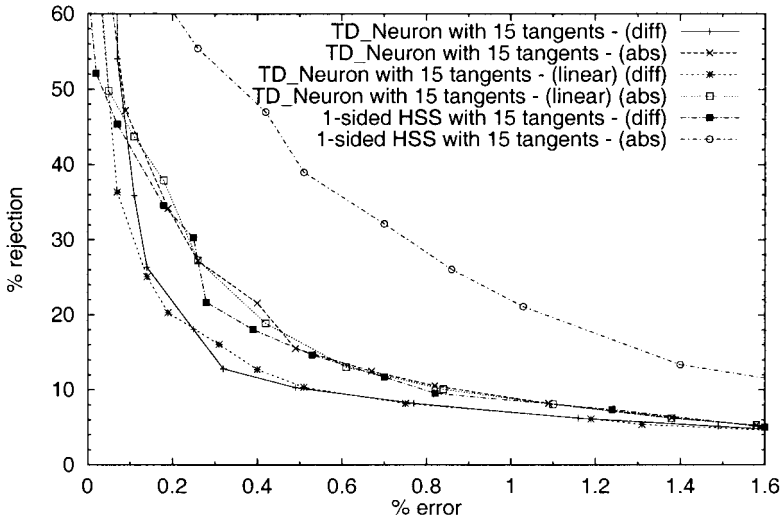


Figure 5: Detail of error-rejection curves of one-sided HSS and TD-Neuron for two different rejection criteria: threshold on the output absolute value (*abs*) and threshold on the difference between the best and the second-best output models (*diff*). The curves for the TD-Neuron model with removed nonlinear output are individuated by the additional label (*linear*).

We also compared the TD-Neuron constructive algorithm to two different versions of the HSS algorithm, the LVQ2.1 algorithm and the 1-NN classification criterion. The obtained results over the NIST-3 database of handwritten digits show that the TD-Neuron is superior to the HSS algorithms based on singular value decomposition and the LVQ algorithms, since it reaches a better trade-off between error and rejection. Moreover, we have assessed that the better trade-off is mainly due to the discriminant capabilities of our model.

Concerning the proposed neural formulation, we believe that the non-linear sigmoidal transformation is useful since removing it and using the inverted target ( $\sigma_k^{-1}(t_k)$ ) for training would drastically reduce the size of the space of solutions in the weight space. In fact, very high or very low values for the net input would not minimize the error function with the inverted target, while being fully acceptable in the proposed model. Moreover, the non-linearity increases the stability of learning because of the saturated output.

During the model generation, for a fixed number of tangent vectors, the HSS algorithm is faster than ours because it needs only a fraction of the training examples (only one class). However, our algorithm is remarkably more efficient than HSS algorithms when a family of tangent models, with an increasing number of tangent vectors, must be generated. Also the LVQ

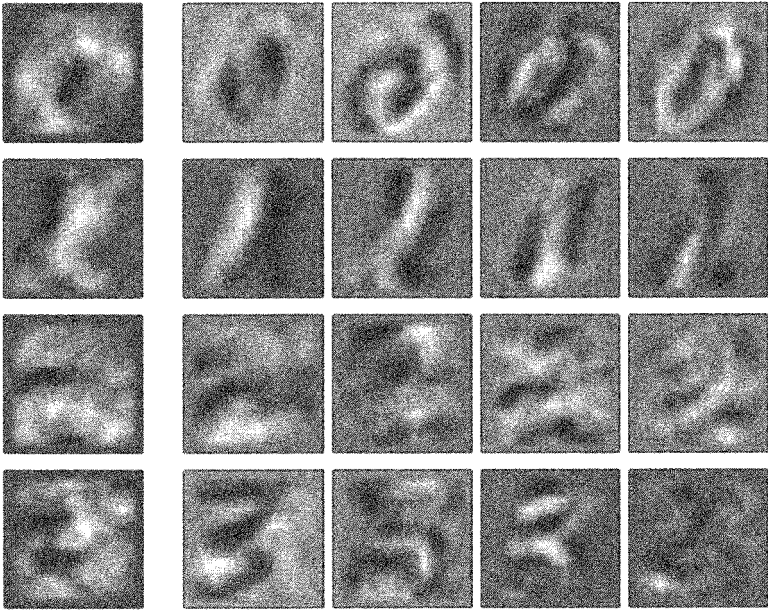


Figure 6: Tangent models obtained by the TD-Neuron for digits 0, 1, 2, and 3. The centroids are shown in the left-most column; the remaining columns show the first four tangent vectors.

algorithms are faster than TD-Neuron, but they have as a drawback a poor rejection performance.

An additional advantage of the TD-Neuron model is that because the training algorithm is based on a gradient-descent technique, several TD-Neurons can be arranged to form a hidden layer in a feedforward network with standard output neurons, which can be trained by a trivial extension of backpropagation. This may lead to a remarkable increase in the transformation-invariant features of the system. Furthermore, it should be possible to extract information easily from the network regarding the most important features used during classification (see Figure 6).

## References

---

- Hastie, T., Simard, P. Y., & Säcker, E. (1995). Learning prototype models for tangent distance. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems*, 7 (pp. 999–1006). Cambridge, MA: MIT Press.
- Hinton, G. E., Dayan, P., & Revow, M. (1997). Modeling the manifold of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8, 65–74.

- Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., & Torkkola, K. (1996). *LVQ\_PAK: The learning vector quantization program package* (Tech. Rep. No. A30). Espoo, Finland: Helsinki University of Technology, Laboratory of Computer and Information Science. Available online at: <http://www.cis.hut.fi/nnrc/nnrcc-programs.html>.
- Schwenk, H., & Milgram, M. (1995a). Learning discriminant tangent models for handwritten character recognition. In *International Conference on Artificial Neural Networks* (pp. 985–988). Berlin: Springer-Verlag.
- Schwenk, H., & Milgram, M. (1995b). Transformation invariant autoassociation with application to handwritten character recognition. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems*, 7 (pp. 991–998). Cambridge, MA: MIT Press.
- Simard, P. Y. (1994). Efficient computation of complex distance metrics using hierarchical filtering. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in neural information processing systems*, 6 (pp. 168–175). San Mateo, CA: Morgan Kaufmann.
- Simard, P. Y., LeCun, Y., & Denker, J. (1993). Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems*, 5 (pp. 50–58). San Mateo, CA: Morgan Kaufmann.
- Sona, D., Sperduti, A., & Starita, A. (1997). A constructive learning algorithm for discriminant tangent models. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems*, 9 (pp. 786–792). Cambridge, MA: MIT Press.
- Sperduti, A., & Stork, D. G. (1995). A rapid graph-based method for arbitrary transformation-invariant pattern classification. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in neural information processing systems*, 7 (pp. 665–672). Cambridge, MA: MIT Press.