# 12 Transformation Invariance in Pattern Recognition – Tangent Distance and Tangent Propagation

Patrice Y. Simard[1], Yann A. LeCun[1], John S. Denker[1], and Bernard Victorri[2]

[1] Image Processing Services Research Lab, AT& T Labs - Research, 100 Schulz Drive, Red Bank, NJ 07701-7033, USA
patrice@research.att.com, http://www.research.att.com/info/patrice
[2] CNRS, ELSAP, ENS, 1 rue Maurice Arnoux, F-92120 MONTROUGE, France.

**Abstract.** In pattern recognition, statistical modeling, or regression, the amount of data is a critical factor affecting the performance. If the amount of data and computational resources are unlimited, even trivial algorithms will converge to the optimal solution. However, in the practical case, given limited data and other resources, satisfactory performance requires sophisticated methods to regularize the problem by introducing *a priori* knowledge. Invariance of the output with respect to certain transformations of the input is a typical example of such *a priori* knowledge. In this chapter, we introduce the concept of tangent vectors, which compactly represent the essence of these transformation invariances, and two classes of algorithms, "tangent distance" and "tangent propagation", which make use of these invariances to improve performance.

## 12.1 Introduction

Pattern Recognition is one of the main tasks of biological information processing systems, and a major challenge of computer science. The problem of pattern recognition is to classify objects into categories, given that objects in a particular category may have widely-varying features, while objects in different categories may have quite similar features. A typical example is handwritten digit recognition. Characters, typically represented as fixed-size images (say 16 by 16 pixels), must be classified into one of 10 categories using a *classification function*. Building such a classification function is a major technological challenge, as irrelevant variabilities among objects of the same class must be eliminated, while meaningful differences between objects of different classes must be identified. These classification functions for most real pattern recognition tasks are too complicated to be synthesized "by hand" using only what humans know about the task. Instead, we use sophisticated techniques that combine humans' *a priori* knowledge with information automatically extracted from a set of labeled examples (the training set). These techniques can be divided into two camps, according to the number of parameters they require: the “memory based” algorithms, which in effect store a sizeable subset of the entire training set, and the “learned-function” techniques, which learn by adjusting a comparatively small number of parameters. This distinction is arbitrary because the patterns stored by a memory-based algorithm can be considered the parameters of a very complex learned function. The distinction is however useful in this work, because memory based algorithms often rely on a metric which can be modified to incorporate transformation invariances, while learned-function algorithms consist of selecting a classification function, the derivatives of which can be constrained to reflect the same transformation invariances. The two methods for incorporating invariances are different enough to justify two independent sections.

### 12.1.1 Memory based algorithms

To compute the classification function, many practical pattern recognition systems, and several biological models, simply store all the examples, together with their labels, in a memory. Each incoming pattern can then be compared to all the stored prototypes, and the labels associated with the prototypes that best match the input determine the output. The above method is the simplest example of the *memory-based* models. Memory-based models require three things: a *distance measure* to compare inputs to prototypes, an *output function* to produce an output by combining the labels of the prototypes, and a *storage scheme* to build the set of prototypes.

All three aspects have been abundantly treated in the literature. Output functions range from simply voting the labels associated with the $k$ closest prototypes (K-Nearest Neighbors), to computing a score for each class as a linear combination of the distances to all the prototypes, using fixed [21] or learned [5] coefficients. Storage schemes vary from storing the entire training set, to picking appropriate subsets of it (see [8], chapter 6, for a survey) to learned-functions such as learning vector quantization (LVQ) [17] and gradient descent. Distance measures can be as simple as the Euclidean distance, assuming the patterns and prototypes are represented as vectors, or more complex as in the generalized quadratic metric [10] or in elastic matching methods [15].

A simple but inefficient pattern recognition method is to use a simple distance measure, such as Euclidean distance between vectors representing the raw input, combined with a very large set of prototypes. This method is inefficient because almost all possible instances of a category must be present in the prototype set. In the case of handwritten digit recognition, this means that digits of each class in all possible positions, sizes, angles, writing styles, line thicknesses, skews, etc... must be stored. In real situations, this approach leads to impractically large prototype sets or to mediocre recognition accuracy as illustrated in Figure 12.1. An unlabeled image of a thick, slanted “9” must be classified by finding the closest prototype image out of two images representing respectively a thin, upright “9” and a thick, slanted“4”. According to the Euclidean distance (sum of the squares of the pixel to pixel differences), the “4” is closer. The result is an incorrect classification. The classical way of dealing with this problem is to

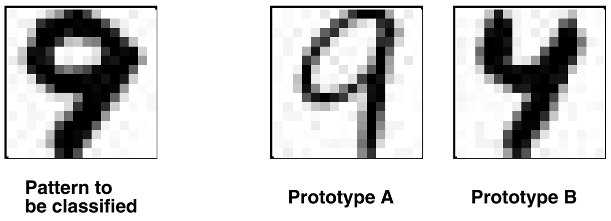

**Fig. 12.1.** According to the Euclidean distance the pattern to be classified is more similar to prototype B. A better distance measure would find that prototype A is closer because it differs mainly by a rotation and a thickness transformation, two transformations which should leave the classification invariant.

use a so-called *feature extractor* whose purpose is to compute a representation of the patterns that is minimally affected by transformations of the patterns that do not modify their category. For character recognition, the representation should be invariant with respect to position, size changes, slight rotations, distortions, or changes in line thickness. The design and implementation of feature extractors is the major bottleneck of building a pattern recognition system. For example, the problem illustrated in Figure 12.1 can be solved by deslanting and thinning the images.

An alternative to this is to use an invariant *distance measure* constructed in such a way that the distance between a prototype and a pattern will not be affected by irrelevant transformations of the pattern or of the prototype. With an invariant distance measure, each prototype can match many possible instances of pattern, thereby greatly reducing the number of prototypes required.

The natural way of doing this is to use "deformable" prototypes. During the matching process, each prototype is deformed so as to best fit the incoming pattern. The quality of the fit, possibly combined with a measure of the amount of deformation, is then used as the distance measure [15]. With the example of Figure 12.1, the "9" prototype would be rotated and thickened so as to best match the incoming "9". This approach has two shortcomings. First, a set of allowed deformations must be designed based on *a priori* knowledge. Fortunately, this is feasible for many tasks, including character recognition. Second, the search for the best-matching deformation is often enormously expensive, and/or unreliable. Consider the case of patterns that can be represented by vectors. For example, the pixel values of a 16 by 16 pixel character image can be viewed as the components of a 256-dimensional vector. One pattern, or one prototype, is a point in this 256-dimensional space. Assuming that the set of allowable transformations is continuous, the set of all the patterns that can be obtained by transforming one prototype using one or a combination of allowable transformations is a surface in the 256-D pixel space. More precisely, when a pattern $P$ is transformed (e.g. rotated) according to a transformation $s(P, \alpha)$ which depends on one parameter $\alpha$ (e.g. the angle of the rotation), the set of all the transformed patterns

$$S_P = \{x \mid \exists \boldsymbol{\alpha} \text{ for which } x = s(P, \alpha)\} \tag{12.1}$$

is a one-dimensional curve in the vector space of the inputs. In the remainder of this chapter, we will always assume that we have chosen $s$ be differentiable with respect to both $P$ and $\alpha$ and such that $s(P, 0) = P$.

When the set of transformations is parameterized by $n$ parameters $\alpha_i$ (rotation, translation, scaling, etc.), the intrinsic dimension of the manifold $S_P$ is at most $n$. For example, if the allowable transformations of character images are horizontal and vertical shifts, rotations, and scaling, the surface will be a 4-dimensional manifold.

In general, the manifold will not be linear. Even a simple image translation corresponds to a highly non-linear transformation in the high-dimensional pixel space. For example, if the image of an "8" is translated upward, some pixels oscillate from white to black and back several times. Matching a deformable prototype to an incoming pattern now amounts to finding the point on the surface that is at a minimum distance from the point representing the incoming pattern. This non-linearity makes the matching much more expensive and unreliable. Simple minimization methods such as gradient descent (or conjugate gradient) can be used to find the minimum-distance point, however, these methods only converge to a *local* minimum. In addition, running such an iterative procedure for each prototype is usually prohibitively expensive.

If the set of transformations happens to be linear in pixel space, then the manifold is a linear subspace (a hyperplane). The matching procedure is then reduced to finding the shortest distance between a point (vector) and a hyperplane, which is an easy-to-solve quadratic minimization problem. This special case has been studied in the statistical literature and is sometimes referred to as Procrustes analysis [24]. It has been applied to signature verification [12] and on-line character recognition [26].

This chapter considers the more general case of non-linear transformations such as geometric transformations of gray-level images. Remember that even a simple image translation corresponds to a highly non-linear transformation in the high-dimensional pixel space. The main idea of the chapter is to approximate the surface of possible transforms of a pattern by its tangent plane at the pattern, thereby reducing the matching to finding the shortest distance between two planes. This distance is called the *tangent distance.* The result of the approximation is shown in Figure 12.2, in the case of rotation for handwritten digits. At the top of the figure, is the theoretical curve in pixel space which represents equation (12.1), together with its linear approximation. Points of the transformation curve are depicted below for various amounts of rotation (each angle corresponds to a value of $\alpha$). The bottom of Figure 12.2 depicts the linear approximation of the curve $s(P, \alpha)$ given by the Taylor expansion of $s$ around $\alpha = 0$:

$$s(P, \alpha) = s(P, 0) + \alpha \frac{\partial s(P, \alpha)}{\partial \alpha} + O(\alpha^2) \approx P + \alpha T. \tag{12.2}$$

This linear approximation is completely characterized by the point $P$ and the tangent vector $T = \frac{\partial s(P,\alpha)}{\partial \alpha}$. Tangent vectors, also called the Lie derivatives of the transformation $s$, will be the subject of section 12.4. As can be seen from

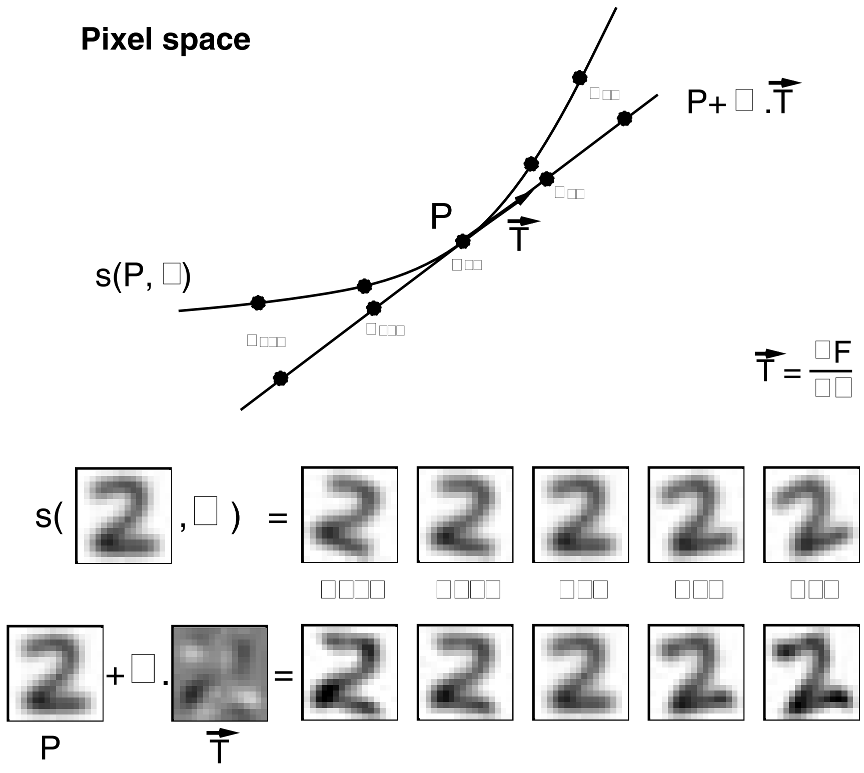

**Fig. 12.2.** Top: Representation of the effect of the rotation in pixel space. Middle: Small rotations of an original digitized image of the digit “2”, for different angle values of $\alpha$. Bottom: Images obtained by moving along the tangent to the transformation curve for the same original digitized image $P$ by adding various amounts ($\alpha$) of the tangent vector $T$.

Figure 12.2, for reasonably small angles ($\|\alpha\| < 1$), the approximation is very good.

Figure 12.3 illustrates the difference between the Euclidean distance, the full invariant distance (minimum distance between manifolds) and the tangent distance. In the figure, both the prototype and the pattern are deformable (two-sided distance), but for simplicity or efficiency reasons, it is also possible to deform only the prototype or only the unknown pattern (one-sided distance).

Although in the following we will concentrate on using tangent distance to recognize images, the method can be applied to many different types of signals: temporal signals, speech, sensor data...

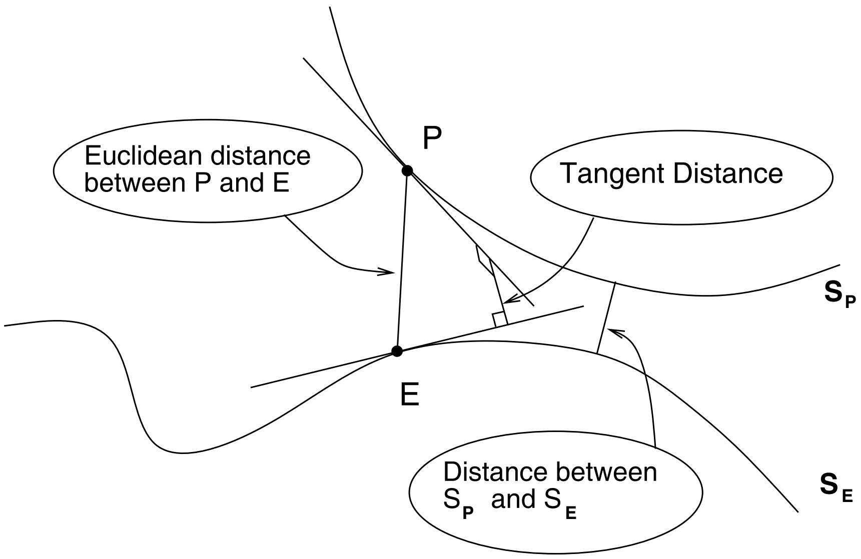
Euclidean distance between P and E
P
Tangent Distance
$S_P$
E
Distance between $S_P$ and $S_E$
$S_E$

**Fig. 12.3.** Illustration of the Euclidean distance and the tangent distance between $P$ and $E$. The curves $S_p$ and $S_e$ represent the sets of points obtained by applying the chosen transformations (for example translations and rotations) to $P$ and $E$. The lines going through $P$ and $E$ represent the tangent to these curves. Assuming that working space has more dimensions than the number of chosen transformations (on the diagram, assume one transformation in a 3-D space) the tangent spaces do not intersect and the tangent distance is uniquely defined.

### 12.1.2 Learned-function algorithms

Rather than trying to keep a representation of the training set, it is also possible to choose a classification function by learning a set of parameters. This is the approach taken in neural networks, curve fitting, regression, et cetera.

We assume all data is drawn independently from a given statistical distribution $\mathcal{P}$, and our learning machine is characterized by the set of functions it can implement, $G_w(x)$, indexed by the vector of parameters $w$. We write $F(x)$ to represent the "correct" or "desired" labeling of the point $x$. The task is to find a value for $w$ such that $G_w$ best approximates $F$. We can use a finite set of training data to help find this vector. We assume the correct labeling $F(x)$ is known for all points in the training set. For example, $G_w$ may be the function computed by a neural net having weights $w$, or $G_w$ may be a polynomial having coefficients $w$. Without additional information, finding a value for $w$ is an ill-posed problem unless the number of parameters is small and/or the size of the training set is large. This is because the training set does not provide enough information to distinguish the best solution among all the candidate $w$s. This problem is illustrated in Figure 12.4 (left). The desired function $F$ (solid line) is to be approximated by a functions $G_w$ (dotted line) from four examples $\{(x_i, F(x_i))\}_{i=1,2,3,4}$. As exemplified in the picture, the fitted function $G_w$

largely disagrees with the desired function $F$ between the examples, but it is not possible to infer this from the training set alone. Many values of $w$ can generate many different functions $G_w$, some of which may be terrible approximations of $F$, even though they are in complete agreement with the training set. Because of this, it is customary to add "regularizers", or additional constraints, to restrict the search of an acceptable $w$. For example, we may require the function $G_w$ to be "smooth", by adding the constraint that $\|w\|^2$ should be minimized. It is important that the regularizer reflects a property of $F$, hence regularizers depend on *a priori* knowledge about the function to be modeled.

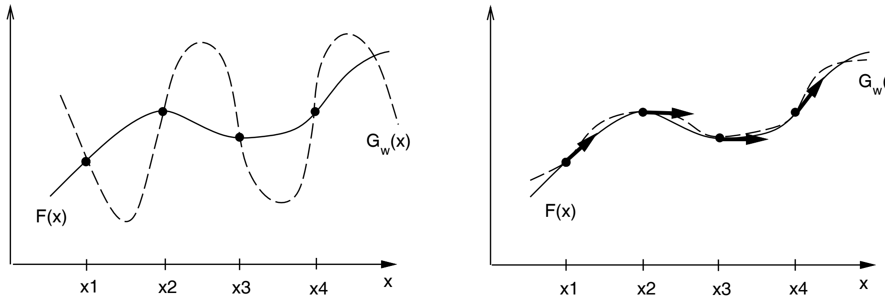

**Fig. 12.4.** Learning a given function (solid line) from a limited set of examples ($x_1$ to $x_4$). The fitted curves are shown by dotted line. Left: The only constraint is that the fitted curve goes through the examples. Right: The fitted curves not only go through each example but also its derivatives evaluated at the examples agree with the derivatives of the given function.

Selecting a good family $\mathcal{G} = \{G_w, w \in \Re^q\}$ of functions is a difficult task, sometimes known as "model selection" [16, 14]. If $\mathcal{G}$ contains a large family of functions, it is more likely that it will contain a good approximation of $F$ (the function we are trying to approximate), but it is also more likely that the selected candidate (using the training set) will generalize poorly because many functions in $\mathcal{G}$ will agree with the training data and take outrageous values between the training samples. If, on the other hand, $\mathcal{G}$ contains a small family of functions, it is more likely that a function $G_w$ which fits the data will be a good approximation of $F$. The capacity of the family of functions $\mathcal{G}$ is often referred to as the VC dimension [28, 27]. If a large amount of data is available, $\mathcal{G}$ should contain a large family of functions (high VC dimension), so that more functions can be approximated, and in particular, $F$. If, on the other hand, the data is scarce, $\mathcal{G}$ should be restricted to a small family of functions (low VC dimension), to control the values between the (more distant) samples[1]. The VC dimension

[1] Note that this point of view also applies to memory based systems. In the case where *all* the training data can be kept in memory, however, the VC dimension is infinite, and the formalism is meaningless. The VC dimension is a learning paradigm and is not useful unless learning is involved.

can also be controlled by putting a knob on how much effect is given to some regularizers. For instance it is possible to control the capacity of a neural network by adding "weight decay" as a regularizer. Weight decay is a heuristic that favors smooth classification functions, by making a tradeoff by decreasing $\|w\|^2$ at the cost, usually, of slightly increased error on the training set. Since the optimal classification function is not necessarily smooth, for instance at a decision boundary, the weight decay regularizer can have adverse effects.

As mentioned earlier, the regularizer should reflect interesting properties (*a priori* knowledge) of the function to be learned. If the functions $F$ and $G_w$ are assumed to be differentiable, which is generally the case, the search for $G_w$ can be greatly improved by requiring that $G_w$'s derivatives evaluated at the points $\{x_i\}$ are more or less equal (this is the regularizer knob) to the derivatives of $F$ at the same points (Figure 12.4 right). This result can be extended to multidimensional inputs. In this case, we can impose the equality of the derivatives of $F$ and $G_w$ in *certain directions*, not necessarily in all directions of the input space. Such constraints find immediate use in traditional pattern recognition problems. It is often the case that *a priori* knowledge is available on how the desired function varies with respect to some transformations of the input. It is straightforward to derive the corresponding constraint on the directional derivatives of the fitted function $G_w$ in the directions of the transformations (previously named tangent vectors). Typical examples can be found in pattern recognition where the desired classification function is known to be invariant with respect to some transformation of the input such as translation, rotation, scaling, etc., in other words, the directional derivatives of the classification function in the directions of these transformations is zero.

This is illustrated in Figure 12.4. The right part of the figure shows how the additional constraints on $G_w$ help generalization by constraining the values of $G_w$ outside the training set. For every transformation which has a known effect on the classification function, a regularizer can be added in the form of a constraint on the directional derivative of $G_w$ in the direction of the tangent vector (such as the one depicted in Figure 12.2), computed from the curve of transformation.

The next section will analyze in detail how to use a distance based on tangent vector in memory based algorithms. The subsequent section will discuss the use of tangent vectors in neural network, with the tangent propagation algorithm. The last section will compare different algorithms to compute tangent vectors.

## 12.2 Tangent Distance

The Euclidean distance between two patterns $P$ and $E$ is in general not appropriate because it is sensitive to irrelevant transformations of $P$ and of $E$. In contrast, the transformed distance $\mathcal{D}(E, P)$ is defined to be the minimal distance between the two manifolds $S_P$ and $S_E$, and is therefore invariant with respect to the transformation used to generate $S_P$ and $S_E$ (see Figure 12.3). Unfortunately, these manifolds have no analytic expression in general, and finding the distance between them is a difficult optimization problem with multiple local minima. Besides, true invariance is not necessarily desirable since a rotation of a "6" into a "9" does not preserve the correct classification.

Our approach consists of computing the minimum distance between the linear surfaces that best approximate the non-linear manifolds $S_P$ and $S_E$. This solves three problems at once: 1) linear manifolds have simple analytical expressions which can be easily computed and stored, 2) finding the minimum distance between linear manifolds is a simple least squares problem which can be solved efficiently and, 3) this distance is locally invariant but not globally invariant. Thus the distance between a "6" and a slightly rotated "6" is small but the distance between a "6" and a "9" is large. The different distances between $P$ and $E$ are represented schematically in Figure 12.3.

The figure represents two patterns $P$ and $E$ in 3-dimensional space. The manifolds generated by $s$ are represented by one-dimensional curves going through $E$ and $P$ respectively. The linear approximations to the manifolds are represented by lines tangent to the curves at $E$ and $P$. These lines do not intersect in 3 dimensions and the shortest distance between them (uniquely defined) is $D(E, P)$. The distance between the two non-linear transformation curves $\mathcal{D}(E, P)$ is also shown on the figure.

An efficient implementation of the tangent distance $D(E, P)$ will be given in the next section, using image recognition as an illustration. We then compare our methods with the best known competing methods. Finally we will discuss possible variations on the tangent distance and how it can be generalized to problems other than pattern recognition.

### 12.2.1 Implementation

In this section we describe formally the computation of the tangent distance. Let the function $s$ transform an image $P$ to $s(P, \alpha)$ according to the parameter $\alpha$. We require $s$ to be differentiable with respect to $\alpha$ and $P$, and require $s(P, 0) = P$. If $P$ is a 2 dimensional image for instance, $s(P, \alpha)$ could be a rotation of $P$ by the angle $\alpha$. If we are interested in all transformations of images which conserve distances (isometry), $s(P, \alpha)$ would be a rotation by $\alpha_\theta$ followed by a translation by $\alpha_x, \alpha_y$ of the image $P$. In this case $\alpha = (\alpha_\theta, \alpha_x, \alpha_y)$ is a vector of parameters of dimension 3. In general, $\alpha = (\alpha_1, \ldots, \alpha_m)$ is of dimension $m$.

Since $s$ is differentiable, the set $S_P = \{x \mid \exists \alpha \text{ for which } x = s(P, \alpha)\}$ is a differentiable manifold which can be approximated to the first order by a hyperplane $T_P$. This hyperplane is tangent to $S_P$ at $P$ and is generated by the columns of matrix

$$L_P = \left. \frac{\partial s(P, \alpha)}{\partial \alpha} \right|_{\alpha = \mathbf{0}} = \left[ \frac{\partial s(P, \alpha)}{\partial \alpha_1}, \ldots, \frac{\partial s(P, \alpha)}{\partial \alpha_m} \right]_{\alpha = \mathbf{0}} \tag{12.3}$$

which are vectors tangent to the manifold. If $E$ and $P$ are two patterns to be compared, the respective tangent planes $T_E$ and $T_P$ can be used to define a new distance $D$ between these two patterns. The tangent distance $D(E, P)$ between

$E$ and $P$ is defined by

$$D(E,P) = \min_{x \in T_E, y \in T_P} \|x - y\|^2. \tag{12.4}$$

The equation of the tangent planes $T_E$ and $T_P$ is given by:

$$E'(\alpha_E) = E + L_E \alpha_E \tag{12.5}$$
$$P'(\alpha_P) = P + L_P \alpha_P \tag{12.6}$$

where $L_E$ and $L_P$ are the matrices containing the tangent vectors (see equation (12.3)) and the vectors $\alpha_E$ and $\alpha_P$ are the coordinates of $E'$ and $P'$ (using bases $L_E$ and $L_P$) in the corresponding tangent planes. Note that $E'$, $E$, $L_E$ and $\alpha_E$ denote vectors and matrices in linear equations (12.5). For example, if the pixel space was of dimension 5, and there were two tangent vectors, we could rewrite equation (12.5) as

$$\begin{bmatrix} E'_1 \\ E'_2 \\ E'_3 \\ E'_4 \\ E'_5 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \end{bmatrix} + \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \\ L_{31} & L_{32} \\ L_{41} & L_{42} \\ L_{51} & L_{52} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}. \tag{12.7}$$

The quantities $L_E$ and $L_P$ are attributes of the patterns so in many cases they can be precomputed and stored.

Computing the tangent distance

$$D(E,P) = \min_{\alpha_E, \alpha_P} \|E'(\alpha_E) - P'(\alpha_P)\|^2 \tag{12.8}$$

amounts to solving a linear least squares problem. The optimality condition is that the partial derivatives of $D(E,P)$ with respect to $\alpha_P$ and $\alpha_E$ should be zero:

$$\frac{\partial D(E,P)}{\partial \alpha_E} = 2(E'(\alpha_E) - P'(\alpha_P))^\top L_E = 0 \tag{12.9}$$
$$\frac{\partial D(E,P)}{\partial \alpha_P} = 2(P'(\alpha_P) - E'(\alpha_E))^\top L_P = 0. \tag{12.10}$$

Substituting $E'$ and $P'$ by their expressions yields to the following linear system of equations, which we must solve for $\alpha_P$ and $\alpha_E$:

$$L_P^\top(E - P - L_P\alpha_P + L_E\alpha_E) = 0 \tag{12.11}$$
$$L_E^\top(E - P - L_P\alpha_P + L_E\alpha_E) = 0. \tag{12.12}$$

The solution of this system is

$$(L_{PE}L_{EE}^{-1}L_E^\top - L_P^\top)(E - P) = (L_{PE}L_{EE}^{-1}L_{EP} - L_{PP})\alpha_P \tag{12.13}$$
$$(L_{EP}L_{PP}^{-1}L_P^\top - L_E^\top)(E - P) = (L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \tag{12.14}$$

where $L_{EE} = L_E^\top L_E$, $L_{PE} = L_P^\top L_E$, $L_{EP} = L_E^\top L_P$ and $L_{PP} = L_P^\top L_P$. LU decompositions of $L_{EE}$ and $L_{PP}$ can be precomputed. The most expensive part in solving this system is evaluating $L_{EP}$ ($L_{PE}$ can be obtained by transposing $L_{EP}$). It requires $m_E \times m_P$ dot products, where $m_E$ is the number of tangent vectors for $E$ and $m_P$ is the number of tangent vectors for $P$. Once $L_{EP}$ has been computed, $\alpha_P$ and $\alpha_E$ can be computed by solving two (small) linear systems of respectively $m_E$ and $m_P$ equations. The tangent distance is obtained by computing $\|E'(\alpha_E) - P'(\alpha_P)\|$ using the value of $\alpha_P$ and $\alpha_E$ in equations (12.5) and (12.6). If $n$ is the dimension of the input space (i.e. the length of vector $E$ and $P$), the algorithm described above requires roughly $n(m_E+1)(m_P+1)+3(m_E^3+m_P^3)$ multiply-adds. Approximations to the tangent distance can however be computed more efficiently.

### 12.2.2 Some illustrative results

**Local Invariance:** The "local[2] invariance" of tangent distance can be illustrated by transforming a reference image by various amounts and measuring its distance to a set of prototypes.

The bottom of Figure 12.5 shows 10 typical handwritten digit images. One of them – the digit "3" – is chosen to be the reference. The reference is translated horizontally by the amount indicated in the abscissa. There are ten curves for Euclidean distance and ten more curves for tangent distance, measuring the distance between the translated reference and one of the 10 digits.

Since the reference was chosen from the 10 digits, it is not surprising that the curve corresponding to the digit "3" goes to 0 when the reference is not translated (0 pixel translation). It is clear from the figure that if the reference (the image "3") is translated by more than 2 pixels, the Euclidean distance will confuse it with other digits, namely "8" or "5". In contrast, there is no possible confusion when tangent distance is used. As a matter of fact, in this example, the tangent distance correctly identifies the reference up to a translation of 5 pixels! Similar curves were obtained with all the other transformations (rotation, scaling, etc...).

The "local" invariance of tangent distance with respect to small transformations generally implies more accurate classification for much larger transformations. This is the single most important feature of tangent distance.

The locality of the invariance has another important benefit: Local invariance can be enforced with *very few* tangent vectors. The reason is that for infinitesimal (local) transformations, there is a direct correspondence[3] between the tangent vectors of the tangent plane and the various compositions of transformations.

[2] Local invariance refers to invariance with respect to small transformations (i.e. a rotation of a very small angle). In contrast, global invariance refers to invariance with respect to arbitrarily large transformations (i.e. a rotation of 180 degrees). Global invariance is not desirable in digit recognition, since we need to distinguish "6" from a "9".

[3] an isomorphism actually, see "Lie algebra" in [6].

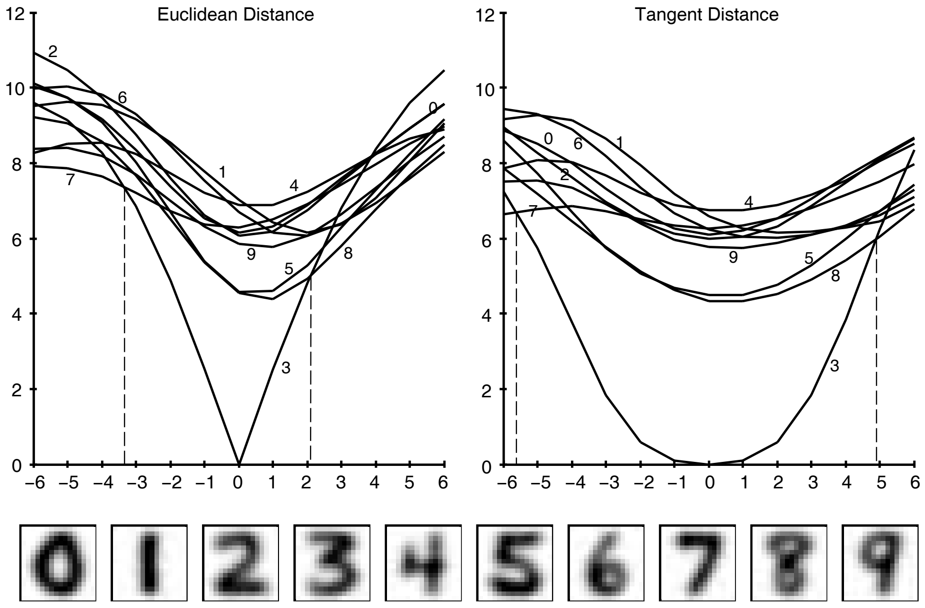

**Fig. 12.5.** Euclidean and tangent distances between 10 typical images of handwritten digits and a translated image of the digit "3". The abscissa represents the amount of horizontal translation (measured in pixels).

For example, the three tangent vectors for X-translation, Y-translation and rotations around the origin, generate a tangent plane corresponding to all the possible compositions of horizontal translations, vertical translations and rotations. The resulting tangent distance is then locally invariant to *all* the translations and *all* the rotations (around any center). Figure 12.6 further illustrates this phenomenon by displaying points in the tangent plane generated from only 5 tangent vectors. Each of these images looks like it has been obtained by applying various combinations of scaling, rotation, horizontal and vertical skewing, and thickening. Yet, the tangent distance between any of these points and the original image is 0.

**Handwritten Digit Recognition:** Experiments were conducted to evaluate the performance of tangent distance for handwritten digit recognition. An interesting characteristic of digit images is that we can readily identify a set of local transformations which do not affect the identity of the character, while covering a large portion of the set of possible *instances* of the character. Seven such image transformations were identified: X and Y translations, rotation, scaling, two hyperbolic transformations (which can generate shearing and squeezing), and line thickening or thinning. The first six transformations were chosen to span the set of all possible linear coordinate transforms in the image plane. (Nevertheless, they correspond to highly non-linear transforms in pixel space.) Additional

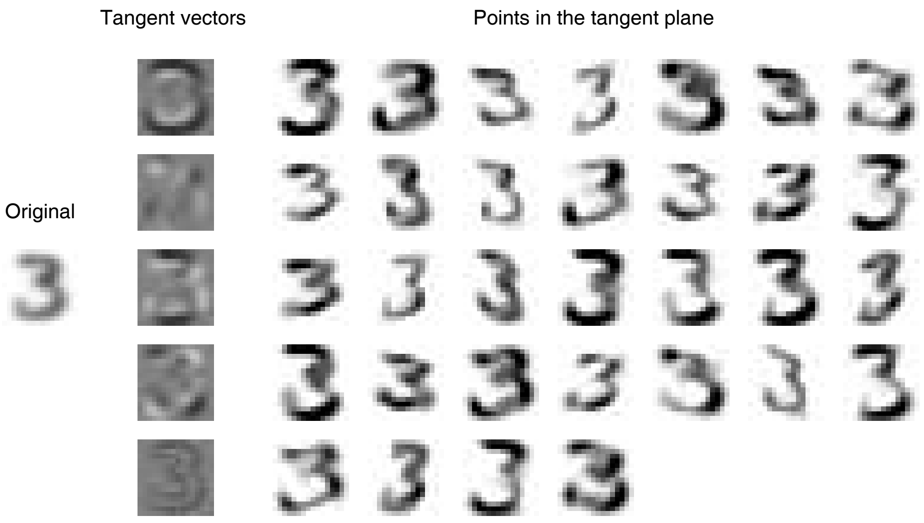

**Fig. 12.6.** Left: Original image. Middle: 5 tangent vectors corresponding respectively to the 5 transformations: scaling, rotation, expansion of the X axis while compressing the Y axis, expansion of the first diagonal while compressing the second diagonal and thickening. Right: 32 points in the tangent space generated by adding or subtracting each of the 5 tangent vectors.

transformations have been tried with less success. Three databases were used to test our algorithm:

**US Postal Service database:** The database consisted of $16 \times 16$ pixel size-normalized images of handwritten digits, coming from US mail envelopes. The training and testing set had respectively 9709 and 2007 examples.

**NIST1 database:** The second experiment was a competition organized by the National Institute of Standards and Technology (NIST) in Spring 1992. The object of the competition was to classify a test set of 59,000 handwritten digits, given a training set of 223,000 patterns.

**NIST2 database:** The third experiment was performed on a database made out of the training and testing database provided by NIST (see above). NIST had divided the data into two sets which unfortunately had different distributions. The training set (223,000 patterns) was easier than the testing set (59,000 patterns). In our experiments we combined these two sets 50/50 to make a training set of 60,000 patterns and testing/validation sets of 10,000 patterns each, all having the same characteristics.

For each of these three databases we tried to evaluate human performance to benchmark the difficulty of the database. For USPS, two members of our group went through the test set and both obtained a 2.5% raw error performance. The human performance on NIST1 was provided by the National Institute of Standard and Technology. The human performance on NIST2 was measured on a small subsample of the database and must therefore be taken with caution. Several of the leading algorithms where tested on each of these databases.

The first experiment used the K-Nearest Neighbor algorithm, using the ordinary Euclidean distance. The prototype set consisted of all available training examples. A 1-Nearest Neighbor rule gave optimal performance in USPS while a 3-Nearest Neighbors rule performed better in NIST2.

The second experiment was similar to the first, but the distance function was changed to tangent distance with 7 transformations. For the USPS and NIST2 databases, the prototype set was constructed as before, but for NIST1 it was constructed by cycling through the training set. Any patterns which were misclassified were added to the prototype set. After a few cycles, no more prototypes are added (the training error was 0). This resulted in 10,000 prototypes. A 3-Nearest Neighbors rule gave optimal performance on this set.

Other algorithms such as neural nets [18, 20], optimal margin classifier [7], local learning [3] and boosting [9] were also used on these databases. A case study can be found in [20].

The results are summarized in Table 12.2.2. As illustrated in the table, the

| | Human | K-NN | T.D. | Lenet1 | Lenet4 | OMC | LL | Boost |
|---|---|---|---|---|---|---|---|---|
| USPS | 2.5 | 5.7 | 2.5 | 4.2 | | 4.3 | 3.3 | 2.6 |
| NIST1 | 1.6 | | 3.2 | | 3.7 | | | 4.1 |
| NIST2 | 0.2 | 2.4 | 1.1 | 1.7 | 1.1 | 1.1 | 1.1 | 0.7 |

**Table 12.1.** Results: Performances in % of errors for (in order) human, K-nearest neighbor, tangent distance, Lenet1 (simple neural network), Lenet4 (large neural network), optimal margin classifier (OMC), local learning (LL) and boosting (Boost).

tangent distance algorithm equals or outperforms all other algorithms we tested, in all cases except one: Boosted LeNet 4 was the winner on the NIST2 database. This is not surprising. The K-nearest neighbor algorithm (with no preprocessing) is very unsophisticated in comparison to local learning, optimal margin classifier, and boosting. The advantange of tangent distance is the *a priori* knowledge of transformation invariance embedded into the distance. When the training data is sufficiently large, as is the case in NIST2, some of this knowledge can be picked up from the data by the more sophisticated algorithms. In other words, the value of *a priori* knowledge decreases as the size of the training set increases.

### 12.2.3 How to make tangent distance work

This section is dedicated to the technological "know how" which is necessary to make tangent distance work with various applications. "Tricks"of this sort are usually not published for various reasons (they are not always theoretically sound, page area is too valuable, the tricks are specific to one particular application, commercial competitive considerations discourage telling everyone how to reproduce the result, etc.), but they are often a determining factor in making the technology a success. Several of these techniques will be discussed here.

**Smoothing the input space:** This is the single most important factor in obtaining good performance with tangent distance. By definition, the tangent vectors are the Lie derivatives of the transformation function $s(P, \alpha)$ with respect to $\alpha$. They can be written as:

$$L_P = \left.\frac{\partial s(P, \alpha)}{\partial \alpha}\right| = \lim_{\epsilon \to 0} \frac{s(P, \epsilon) - s(P, 0)}{\epsilon}. \qquad (12.15)$$

It is therefore very important that $s$ be differentiable (and well behaved) with respect to $\alpha$. In particular, it is clear from equation (12.15) that $s(P, \epsilon)$ must be computed for $\epsilon$ arbitrarily small. Fortunately, even when $P$ can only take discrete values, it is easy to make $s$ differentiable. The trick is to use a smoothing interpolating function $C_\sigma$ as a preprocessing for $P$, such that $s(C_\sigma(P), \alpha)$ is differentiable (with respect to $C_\sigma(P)$ and $\alpha$, not with respect to $P$). For instance, if the input space for $P$ is binary images, $C_\sigma(P)$ can be a convolution of $P$ with a Gaussian function of standard deviation $\sigma$. If $s(C_\sigma(P), \alpha)$ is a translation of $\alpha$ pixels, the derivative of $s(C_\sigma(P), \alpha)$ can easily be computed since $s(C_\sigma(P), \epsilon)$ can be obtained by translating Gaussian functions. This preprocessing will be discussed in more details in section 12.4.

The smoothing factor $\sigma$ controls the locality of the invariance. The smoother the transformation curve defined by $s$ is, the longer the linear approximation will be valid. In general the best smoothing is the maximum smoothing which does not blur the features. For example, in handwritten character recognition with 16x16 pixel images, a Gaussian function with a standard deviation of 1 pixel yielded the best results. Increased smoothing led to confusion (such as a "5" mistaken for "6" because the lower loop had been closed by the smoothing) and decreased smoothing didn't make full use of the invariance properties.

If the available computation time allows it, the best strategy is to extract features first, smooth shamelessly, and then compute the tangent distance on the smoothed features.

**Controlled deformation:** The linear system given in equation (12.8) is singular if some of the tangent vectors for $E$ or $P$ are parallel. Although the probability of this happening is zero when the data is taken from a real-valued continuous distribution (as is the case in handwritten character recognition), it is possible that a pattern may be duplicated in both the training and the test set, resulting in a division by zero error. The fix is quite simple and elegant. Equation (12.8) can be replaced by equation:

$$D(E, P) = \min_{\alpha_E, \alpha_P} \|E + L_E \alpha_E - P - L_P \alpha_P\|^2 + k\|L_E \alpha_E\|^2 + k\|L_P \alpha_P\|^2. \quad (12.16)$$

The physical interpretation of this equation, depicted in Figure 12.7, is that the point $E'(\alpha_E)$ on the tangent plane $T_E$ is attached to $E$ with a spring with

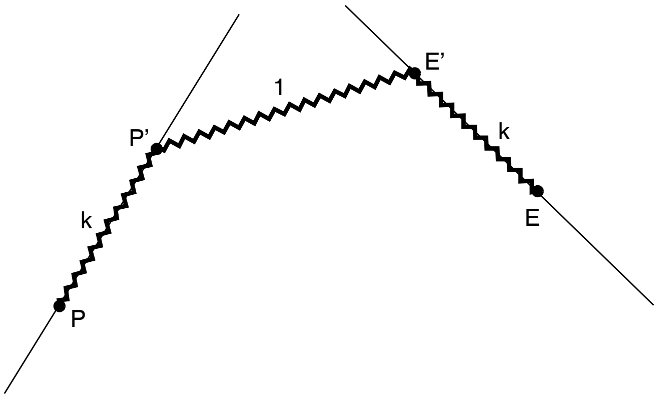

**Fig. 12.7.** The tangent distance between $E$ and $P$ is the elastic energy stored in each of the three springs connecting $P$, $P'$, $E'$ and $E$. $P'$ and $E'$ can move without friction along the tangent planes. The spring constants are indicated on the figure.

spring constant $k$ and to $P'(\alpha_p)$ (on the tangent plane $T_P$) with spring constant 1, and $P'(\alpha_p)$ is also attached to $P$ with spring constant $k$. (All three springs have zero natural length.) The new tangent distance is the total potential elastic energy stored of all three springs at equilibrium. As for the standard tangent distance, the solution can easily be obtained by differentiating equation (12.16) with respect to $\alpha_E$ and $\alpha_P$. The differentiation yields:

$$L_P^\top(E - P - L_P(1+k)\alpha_P + L_E\alpha_E) = 0 \tag{12.17}$$
$$L_E^\top(E - P - L_P\alpha_P + L_E(1+k)\alpha_E) = 0. \tag{12.18}$$

The solution of this system is

$$(L_{PE}L_{EE}^{-1}L_E^\top - (1+k)L_P^\top)(E-P) = (L_{PE}L_{EE}^{-1}L_{EP} - (1+k)^2L_{PP})\alpha_P \tag{12.19}$$
$$(L_{EP}L_{PP}^{-1}L_P^\top - (1+k)L_E^\top)(E-P) = ((1+k)^2L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \tag{12.20}$$

where $L_{EE} = L_E^\top L_E$, $L_{PE} = L_P^\top L_E$, $L_{EP} = L_E^\top L_P$ and $L_{PP} = L_P^\top L_P$. The system has the same complexity as the vanilla tangent distance except that, it always has a solution for $k \geq 0$, and is more numerically stable. Note that for $k = 0$, it is equivalent to the standard tangent distance, while for $k = \infty$, we have the Euclidean distance. This approach is also very useful when the number of tangent vectors is greater or equal than the number of dimensions of the space. The standard tangent distance would most likely be zero (when the tangent spaces intersect), but the "spring" tangent distance still expresses valuable information about the invariances.

If the number of dimension of the input space is large compared to the number of tangent vectors, keeping $k$ as small as possible is better because it doesn't interfere with the "sliding" along the tangent plane ($E'$ and $P'$ are less constrained).

Contrary to intuition, there is no danger of sliding too far in high dimensional space because tangent vectors are always roughly orthogonal and they could only slide far if they were parallel.

**Hierarchy of distances:** If several invariances are used, classification using tangent distance alone would be quite expensive. Fortunately, if a typical memory based algorithm is used, for example, K-nearest neighbors, it is quite unnecessary to compute the full tangent distance between the unclassified pattern and all the labeled samples. In particular, if a crude estimate of the tangent distance indicates with a sufficient confidence that a sample is very far from the pattern to be classified, no more computation is needed to know that this sample is not one of the K-nearest neighbors. Based on this observation one can build a hierarchy of distances which can greatly reduce the computation of each classification. Let's assume, for instance, that we have $m$ approximations $D_i$ of the tangent distance, ordered such that $D_1$ is the crudest approximation of the tangent distance and $D_m$ is exactly tangent distance (for instance $D_1$ to $D_5$ could be the Euclidean distance with increasing resolution, and $D_6$ to $D_{10}$ each add a tangent vector at full resolution).

The basic idea is to keep a pool of all the prototypes which could potentially be the K-nearest neighbors of the unclassified pattern. Initially the pool contains all the samples. Each of the distances $D_i$ corresponds to a stage of the classification process. The classification algorithm has 3 steps at each stage, and proceeds from stage 1 to stage $m$ or until the classification is complete: Step 1: the distance $D_i$ between all the samples in the pool and the unclassified pattern is computed. Step 2: A classification and a confidence score is computed with these distances. If the confidence is good enough, let's say better than $C_i$ (for instance, if all the samples left in the pool are in the same class) the classification is complete, otherwise proceed to step 3. Step 3: The $K_i$ closest samples, according to distance $D_i$ are kept in the pool, while the remaining samples are discarded.

Finding the $K_i$ closest samples can be done in $O(p)$ (where $p$ is the number of samples in the pool) since these elements need not to be sorted [22, 2]. The reduced pool is then passed to stage $i + 1$.

The two constants $C_i$ and $K_i$ must be determined in advance using a validation set. This can easily be done graphically by plotting the error as a function of $K_i$ and $C_i$ at each stage (starting with all $K_i$ equal to the number of labeled samples and $C_i = 1$ for all stages). At each stage there is a minimum $K_i$ and minimum $C_i$ which give optimal performance on the validation set. By taking larger values, we can decrease the probability of making errors on the test sets. The slightly worse performance of using a hierarchy of distances is often well worth the speed-up. The computational cost of a pattern classification is then equal to:

$$\text{computational cost} \approx \sum_i \begin{array}{l}\text{number of}\\ \text{prototypes}\\ \text{at stage } i\end{array} \times \begin{array}{l}\text{distance}\\ \text{complexity}\\ \text{at stage } i\end{array} \times \begin{array}{l}\text{probability}\\ \text{to reach}\\ \text{stage } i\end{array} \qquad (12.21)$$

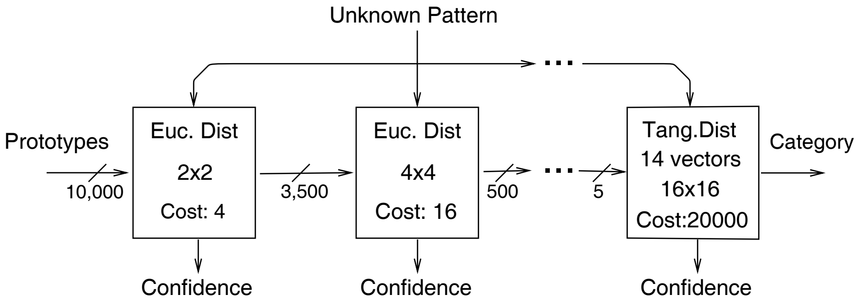

**Fig. 12.8.** Pattern recognition using a hierarchy of distances. The filter proceeds from left (starting with the whole database) to right (where only a few prototypes remain). At each stage distances between prototypes and the unknown pattern are computed and sorted; then the best candidate prototypes are selected for the next stage. As the complexity of the distance increases, the number of prototypes decreases, making computation feasible. At each stage a classification is attempted and a confidence score is computed. If the confidence score is high enough, the remaining stages are skipped.

All this is better illustrated with an example as in Figure 12.8. This system was used for the USPS experiment described in a previous section. In classification of handwritten digits (16x16 pixel images), $D_1$, $D_2$, and $D_3$, were the Euclidean distances at resolution $2 \times 2$, $4 \times 4$ and $8 \times 8$ respectively. $D_4$ was the one sided tangent distance with X-translation, on the sample side only, at resolution $8 \times 8$. $D_5$ was the double sided tangent distance with X-translation at resolution $16 \times 16$. Each of the subsequent distances added one tangent vector on each side (Y-translation, scaling, rotation, hyperbolic deformation1, hyperbolic deformation2 and thickness) until the full tangent distance was computed ($D_{11}$).

Table 12.2 shows the expected number of multiply-adds at each of the stages. It should be noted that the full tangent distance need only be computed for 1 in 20 unknown patterns (probability 0.05), and only with 5 samples out of the original $10,000$. The net speed up was in the order of 500, compared with computing the full tangent distance between every unknown pattern and every sample (this is 6 times faster than computing the Euclidean distance at full resolution).

**Multiple iterations:** Tangent distance can be viewed as one iteration of a Newton-type algorithm which finds the points of minimum distance on the true transformation manifolds. The vectors $\alpha_E$ and $\alpha_P$ are the coordinates of the two closest points in the respective tangent spaces, but they can also be interpreted as the value for the real (non-linear) transformations. In other words, we can use $\alpha_E$ and $\alpha_P$ to compute the points $s(E, \alpha_E)$ and $s(P, \alpha_P)$, the real non-linear transformation of $E$ and $P$. From these new points, we can recompute the tangent vectors, and the tangent distance and reiterate the process. If the appropriate conditions are met, this process can converge to a local minimum in the distance between the two transformation manifolds of $P$ and $E$.

| $i$ | # of T.V. | Reso | # of proto ($K_i$) | # of prod | Probab | # of mul/add |
|---|---|---|---|---|---|---|
| 1 | 0 | 4 | 9709 | 1 | 1.00 | 40,000 |
| 2 | 0 | 16 | 3500 | 1 | 1.00 | 56,000 |
| 3 | 0 | 64 | 500 | 1 | 1.00 | 32,000 |
| 4 | 1 | 64 | 125 | 2 | 0.90 | 14,000 |
| 4 | 2 | 256 | 50 | 5 | 0.60 | 40,000 |
| 6 | 4 | 256 | 45 | 7 | 0.40 | 32,000 |
| 7 | 6 | 256 | 25 | 9 | 0.20 | 11,000 |
| 8 | 8 | 256 | 15 | 11 | 0.10 | 4,000 |
| 9 | 10 | 256 | 10 | 13 | 0.10 | 3,000 |
| 10 | 12 | 256 | 5 | 15 | 0.05 | 1,000 |
| 11 | 14 | 256 | 5 | 17 | 0.05 | 1,000 |

**Table 12.2.** Summary computation for the classification of 1 pattern: The first column is the distance index, the second column indicates the number of tangent vectors (0 for the Euclidean distance), and the third column indicates the resolution in pixels, the fourth is $K_i$ or the number of prototypes on which the distance $D_i$ must be computed, the fifth column indicates the number of additional dot products which must be computed to evaluate distance $D_i$, the sixth column indicates the probability of not skipping that stage after the confidence score has been used, and the last column indicates the total average number of multiply-adds which must be performed (product of column 3 to 6) at each stage.

This process did not improve handwritten character recognition, but it yielded impressive results in face recognition [29]. In that case, each successive iteration was done at increasing resolution (hence combining hierarchical distances and multiple iterations), making the whole process computationally efficient.

## 12.3 Tangent Propagation

The previous section dealt with memory-based techniques. We now apply tangent-distance principles to learned-function techniques.

The key idea is to incorporate the invariance directly into the learned classification function. In this section, we present an algorithm, called "tangent propagation", in which gradient descent is used to propagate information about the invariances of the training data. The process is a generalization of the widely-used "back propagation" method, which propagates information about the training data itself.

We again assume all data is drawn independently from a given statistical distribution $\mathcal{P}$, and our learning machine is characterized by the set of functions in can implement, $G_w(x)$, indexed by the vector of parameters $w$. Ideally, we would like to find $w$ which minimizes the energy function

$$\mathcal{E} = \int \|G_w(x) - F(x)\|^2 d\mathcal{P}(x) \tag{12.22}$$

where $F(x)$ represents the "correct" or "desired" labeling of the point $x$. In the real world we must estimate this integral using only a finite set of training points $B$ drawn the distribution $\mathcal{P}$. That is, we try to minimize

$$E_p = \sum_{i=1}^{p} \|G_w(x_i) - F(x_i)\| \tag{12.23}$$

where the sum runs over the training set $B$. An estimate of $w$ can be computed by following a gradient descent using the weight-update rule:

$$\Delta w = -\eta \frac{\partial E_p}{\partial w}. \tag{12.24}$$

Let's consider an input transformation $s(x, \alpha)$ controlled by a parameter $\alpha$. As always, we require that $s$ is differentiable and that $s(x, 0) = x$. Now, in addition to the known labels of the training data, we assume that $\frac{\partial F(s(x_i,\alpha))}{\partial \alpha}$ is known at $\alpha = 0$ for each point $x$ in the training set. To incorporate the invariance property into $G_w(x)$, we add that the following constraint on the derivative:

$$E_r = \sum_{i=1}^{p} \left| \frac{\partial G_w(s(x_i, \alpha))}{\partial \alpha} - \frac{\partial F(s(x_i, \alpha))}{\partial \alpha} \right|^2_{\alpha=0} \tag{12.25}$$

should be small at $\alpha = 0$. In many pattern classification problems, we are interested in the local classification invariance property for $F(x)$ with respect to the transformation $s$ (the classification does not change when the input is slightly transformed), so we can simplify equation (12.25) to:

$$E_r = \sum_{i=1}^{p} \left| \frac{\partial G_w(s(x_i,\alpha))}{\partial \alpha} \right|^2_{\alpha=0} \tag{12.26}$$

since $\frac{\partial F(s(x_i,\alpha))}{\partial \alpha} = 0$. To minimize this term we can modify the gradient descent rule to use the energy function

$$E = \eta E_p + \mu E_r \tag{12.27}$$

with the weight update rule:

$$\Delta w = -\frac{\partial E}{\partial w}. \tag{12.28}$$

The learning rates (or regularization parameters) $\eta$ and $\mu$ are tremendously important, because they determine the tradeoff between learning the invariances (based on the chosen directional derivatives) versus learning the label itself (i.e. the zeroth derivative) at each point in the training set.

The local variation of the classification function, which appears in equation (12.26) can be written as:

$$\left.\frac{\partial G_w(s(x,\alpha))}{\partial \alpha}\right|_{\alpha=0} = \left.\frac{\partial G_w(s(x,\alpha))}{\partial s(x,\alpha)} \frac{\partial s(x,\alpha)}{\partial \alpha}\right|_{\alpha=0} = \nabla_x G_w(x). \left.\frac{\partial s(x,\alpha)}{\partial \alpha}\right|_{\alpha=0} \tag{12.29}$$

since $s(x,\alpha) = x$ if $\alpha = 0$ and where $\nabla_x G_w(x)$ is the Jacobian of $G_w(x)$ for pattern $x$, and $\partial s(\alpha, x)/\partial \alpha$ is the *tangent vector* associated with transformation $s$ as described in the previous section. Multiplying the tangent vector by the Jacobian involves one forward propagation through a "linearized" version of the network. If $\alpha$ is multi-dimensional, the forward propagation must be repeated for each tangent vector.

The theory of Lie algebras [11] ensures that compositions of local (small) transformations correspond to linear combinations of the corresponding tangent vectors (this result will be discussed further in section 12.4). Consequently, if $E_r(x) = 0$ is verified, the network derivative in the direction of a linear combination of the tangent vectors is equal to the same linear combination of the desired derivatives. In other words, if the network is successfully trained to be locally invariant with respect to, say, horizontal translations and vertical translations, it will be invariant with respect to compositions thereof.

It is possible to devise an efficient algorithm, "tangent prop", for performing the weight update (equation (12.28)). It is analogous to ordinary backpropagation, but in addition to propagating neuron activations, it also propagates the tangent vectors. The equations can be easily derived from Figure 12.9.

### 12.3.1 Local rule

The forward propagation equation is:

$$a_i^l = \sum_j w_{ij}^l x_j^{l-1} \qquad x_i^l = \sigma(a_i^l) \tag{12.30}$$

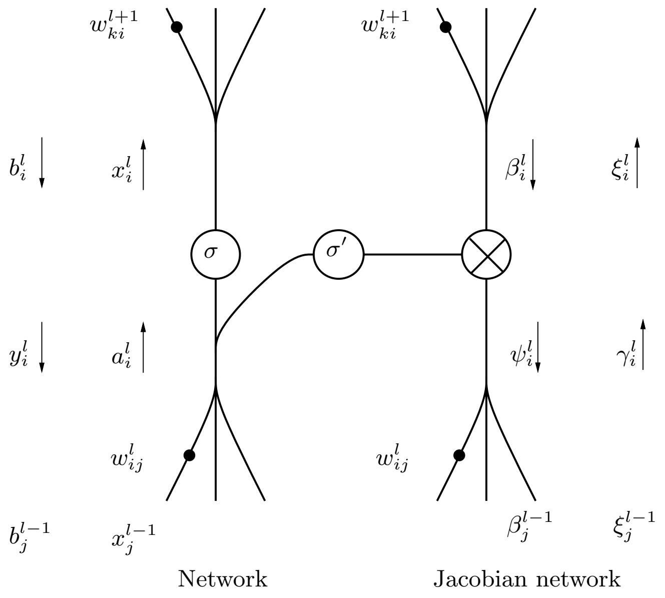

**Fig. 12.9.** Forward propagated variables $(a, x, \gamma, \xi)$, and backward propagated variables $(b, y, \beta, \psi)$ in the regular network (roman symbols) and the Jacobian (linearized) network (greek symbols). Converging forks (in the direction in which the signal is traveling) are sums, diverging forks just duplicate the values.

where $\sigma$ is a non linear differentiable function (typically a sigmoid). The forward propagation starts at the first layer ($l = 1$), with $x^0$ being the input layer, and ends at the output layer ($l = L$). Similarly, The tangent forward propagation (tangent prop) is defined by:

$$\gamma_i^l = \sum_j w_{ij}^l \xi_j^{l-1} \qquad \xi_i^l = \sigma'(a_i^l)\gamma_i^l. \tag{12.31}$$

The tangent forward propagation starts at the first layer ($l = 1$), with $\xi^0$ being the tangent vector $\frac{\partial s(x,\alpha)}{\partial \alpha}$, and ends at the output layer ($l = L$). The tangent gradient backpropagation can be computed using the chain rule:

$$\frac{\partial E}{\partial \xi_i^l} = \sum_k \frac{\partial E}{\partial \gamma_k^{l+1}} \frac{\partial \gamma_k^{l+1}}{\partial \xi_i^l} \qquad \frac{\partial E}{\partial \gamma_i^l} = \frac{\partial E}{\partial \xi_i^l} \frac{\partial \xi_i^l}{\partial \gamma_i^l} \tag{12.32}$$

$$\beta_i^l = \sum_k \psi_k^{l+1} w_{ki}^{l+1} \qquad \psi_i^l = \beta_i^l \sigma'(a_i^l). \tag{12.33}$$

The tangent backward propagation starts at the output layer ($l = L$), with $\xi^L$ being the network variation $\frac{\partial G_w(s(x,\alpha))}{\partial \alpha}$, and ends at the input layer. Similarly, the gradient backpropagation equation is:

$$\frac{\partial E}{\partial x_i^l} = \sum_k \frac{\partial E}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial x_i^l} \qquad \frac{\partial E}{\partial a_i^l} = \frac{\partial E}{\partial x_i^l} \frac{\partial x_i^l}{\partial a_i^l} + \frac{\partial E}{\partial \xi_i^l} \frac{\partial \xi_i^l}{\partial a_i^l} \tag{12.34}$$

$$b_i^l = \sum_k y_k^{l+1} w_{ki}^{l+1} \qquad y_i^l = b_i^l \sigma'(a_i^l) + \beta_i \sigma''(a_i^l) \gamma_i^l. \tag{12.35}$$

The standard backward propagation starts at the output layer ($l = L$), with $x^L = G_w(x^0)$ being the network output, and ends at the input layer. Finally, the weight update is:

$$\Delta w_{ij}^l = -\frac{\partial E}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{ij}^l} - \frac{\partial E}{\partial \gamma_i^l} \frac{\partial \gamma_i^l}{\partial w_{ij}^l} \tag{12.36}$$

$$\Delta w_{ij}^l = -y_i^l x_j^{l-1} - \psi_i^l \xi_j^{l-1}. \tag{12.37}$$

The computation requires one forward propagation and one backward propagation per pattern and per tangent vector during training. After the network is trained, it is approximately locally invariant with respect to the chosen transformation. After training, the evaluation of the learned function is in all ways identical to a network which is not trained for invariance (except that the weights have different values).

### 12.3.2 Results

Two experiments illustrate the advantages of tangent prop. The first experiment is a classification task, using a small (linearly separable) set of 480 binary images of handwritten digits. The training sets consist of 10, 20, 40, 80, 160 or 320 patterns, and the test set contains 160 patterns. The patterns are smoothed using a Gaussian kernel with standard deviation of one half pixel. For each of the training set patterns, the tangent vectors for horizontal and vertical translation are computed. The network has two hidden layers with locally connected shared weights, and one output layer with 10 units (5194 connections, 1060 free parameters) [19]. The generalization performance as a function of the training set size for traditional backprop and tangent prop are compared in Figure 12.10. We have conducted additional experiments in which we implemented not only translations but also rotations, expansions and hyperbolic deformations. This set of 6 generators is a basis for all linear transformations of coordinates for two dimensional images. It is straightforward to implement other generators including gray-level-shifting, "smooth" segmentation, local continuous coordinate transformations and independent image segment transformations.

The next experiment is designed to show that in applications where data is highly correlated, tangent prop yields a large speed advantage. Since the distortion model implies adding lots of highly correlated data, the advantage of tangent prop over the distortion model becomes clear.

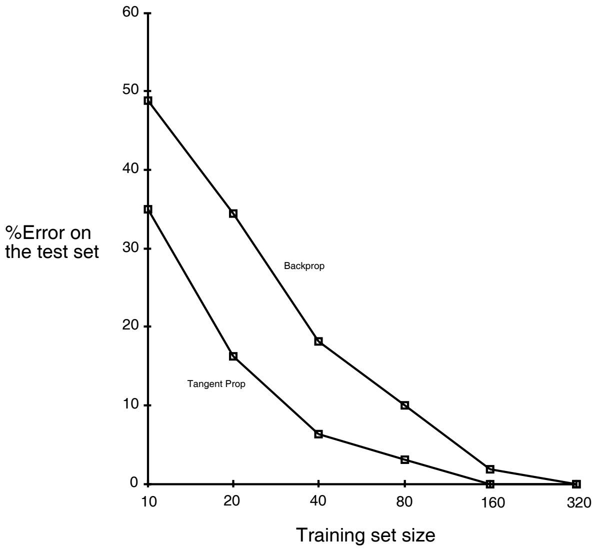

**Fig. 12.10.** Generalization performance curve as a function of the training set size for the tangent prop and the backprop algorithms

The task is to approximate a function that has plateaus at three locations. We want to enforce local invariance near each of the training points (Figure 12.11, bottom). The network has one input unit, 20 hidden units and one output unit. Two strategies are possible: either generate a small set of training points covering each of the plateaus (open squares on Figure 12.11 bottom), or generate one training point for each plateau (closed squares), and enforce local invariance around them (by setting the desired derivative to 0). The training set of the former method is used as a measure of performance for both methods. All parameters were adjusted for approximately optimal performance in all cases. The learning curves for both models are shown in Figure 12.11 (top). Each sweep through the training set for tangent prop is a little faster since it requires only 6 forward propagations, while it requires 9 in the distortion model. As can be seen, stable performance is achieved after 1300 sweeps for the tangent prop, versus 8000 for the distortion model. The overall speedup is therefore about 10.

Tangent prop in this example can take advantage of a very large regularization term. The distortion model is at a disadvantage because the only parameter that effectively controls the amount of regularization is the magnitude of the distortions, and this cannot be increased to large values because the right answer is only invariant under *small* distortions.

### 12.3.3 How to make tangent prop work

**Large network capacity:** Relatively few experiments have been done with tangent propagation. It is clear, however, that the invariance constraint can be

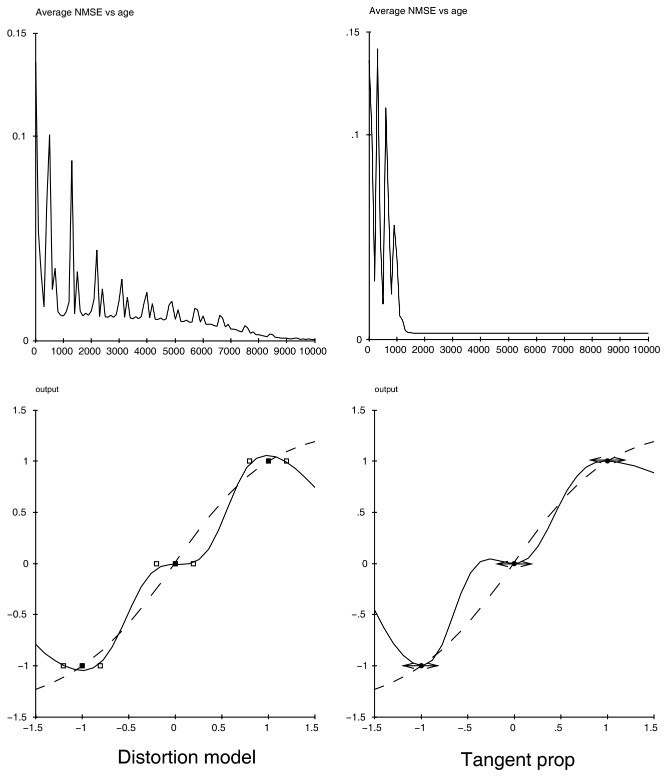

**Fig. 12.11.** Comparison of the distortion model (left column) and tangent prop (right column). The top row gives the learning curves (error versus number of sweeps through the training set). The bottom row gives the final input-output function of the network; the dashed line is the result for unadorned back prop.

extremely beneficial. If the network does not have enough capacity, it will not benefit from the extra knowledge introduced by the invariance.

**Interleaving of the tangent vectors:** Since the tangent vectors introduce even more correlation inside the training set, a substantial speed up can be obtained by alternating a regular forward and backward propagation with a tangent forward and backward propagation (even if there are several tangent vectors, only one is used at each pattern). For instance, if there were 3 tangent vectors, the training sequence could be:

$$x_1, t_1(x_1), x_2, t_2(x_2), x_3, t_3(x_3), x_4, t_1(x_4), x_5, t_2(x_5), \ldots \tag{12.38}$$

where $x_i$ means a forward and backward propagation for pattern $i$ and $t_j(x_i)$ means a tangent forward and backward propagation of tangent vector $j$ of pattern $i$. With such interleaving, the learning converges faster than grouping all the tangent vectors together. Of course, this only makes sense with on-line updates as opposed to batch updates.

## 12.4 Tangent Vectors

In this section, we consider the general paradigm for transformation invariance and for the tangent vectors which have been used in the two previous sections. Before we introduce each transformation and their corresponding tangent vectors, a brief explanation is given of the theory behind the practice. There are two aspects to the problem. First it is possible to establish a formal connection between groups of transformations of the input space (such as translation, rotation, etc. of $\Re^2$) and their effect on a functional of that space (such as a mapping of $\Re^2$ to $\Re$, which may represent an image, in continuous form). The theory of Lie groups and Lie algebra [6] allows us to do this. The second problem has to do with coding. Computer images are finite vectors of discrete variables. How can a theory which was developed for differentiable functionals of $\Re^2$ to $\Re$ be applied to these vectors? We first give a brief explanation of the theorems of Lie groups and Lie algebras which are applicable to pattern recognition. Next, we explore solutions to the coding problem. Finally some examples of transformation and coding are given for particular applications.

### 12.4.1 Lie groups and Lie algebras

Consider an input space $\mathcal{I}$ (the plane $\Re^2$ for example) and a differentiable function $f$ which maps points of $\mathcal{I}$ to $\Re$.

$$f : X \in I \longmapsto f(X) \in \Re. \tag{12.39}$$

The function $f(X) = f(x, y)$ can be interpreted as the continuous (defined for all points of $\Re^2$) equivalent of the discrete computer image $P[i, j]$.

Next, consider a family of transformations $t_\alpha$, parameterized by $\alpha$, which maps bijectively a point of $\mathcal{I}$ to a point of $\mathcal{I}$

$$t_\alpha : X \in \mathcal{I} \longmapsto t_\alpha(X) \in \mathcal{I}. \tag{12.40}$$

We assume that $t_\alpha$ is differentiable with respect to $\alpha$ and $X$, and that $t_0$ is the identity. For example $t_\alpha$ could be the group of affine transformations of $\Re^2$:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha_1 x + \alpha_2 y + \alpha_5 \\ \alpha_3 x + y + \alpha_4 y + \alpha_6 \end{pmatrix} \quad \text{with} \quad \begin{vmatrix} 1 + \alpha_1 & \alpha_2 \\ \alpha_3 & 1 + \alpha_4 \end{vmatrix} \neq 0. \tag{12.41}$$

This is a Lie group[4] with 6 parameters. Another example is the group of direct isometry:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x \cos\theta - y \sin\theta + a \\ x \sin\theta + y \cos\theta + b \end{pmatrix} \tag{12.42}$$

[4] A Lie group is a group that is also a differentiable manifold such that the differentiable structure is compatible with the group structure.

which is a Lie group with 3 parameters.

We now consider the functional $s(f,\alpha)$, defined by

$$s(f,\alpha) = f \circ t_\alpha^{-1}. \tag{12.43}$$

This functional $s$, which takes another functional $f$ as an argument, should remind the reader of Figure 12.2 where $P$, the discrete equivalent of $f$, is the argument of $s$.

The Lie algebra associated with the action of $t_\alpha$ on $f$ is the space generated by the $m$ local transformations $L_{\alpha_i}$ of $f$ defined by:

$$L_{a_i}(f) = \left.\frac{\partial s(f,\alpha)}{\partial \alpha_i}\right|_{\alpha=0}. \tag{12.44}$$

We can now write the local approximation of $s$ as:

$$s(f,\alpha) = f + \alpha_1 L_{\alpha_1}(f) + \alpha_2 L_{\alpha_2}(f) + \ldots + \alpha_m L_{\alpha_m}(f) + o(\|\alpha\|^2)(f). \tag{12.45}$$

This equation is the continuous equivalent of equation (12.2) used in the introduction.

The following example illustrates how $L_{\alpha_i}$ can be computed from $t_\alpha$. Let's consider the group of direct isometry defined in equation (12.42) (with parameter $\alpha = (\theta, a, b)$ as before, and $X = (x,y)$)

$$s(f,\alpha)(X) = f((x-a)\cos\theta + (y-b)\sin\theta, -(x-a)\sin\theta + (y-b)\cos\theta). \tag{12.46}$$

If we differentiate around $\alpha = (0,0,0)$ with respect to $\theta$, we obtain

$$\frac{\partial s(f,\alpha)}{\partial \theta}(X) = y\frac{\partial f}{\partial x}(x,y) + (-x)\frac{\partial f}{\partial y}(x,y) \tag{12.47}$$

that is

$$L_\theta = y\frac{\partial}{\partial x} + (-x)\frac{\partial}{\partial y}. \tag{12.48}$$

The transformation $L_a = -\frac{\partial}{\partial x}$ and $L_b = -\frac{\partial}{\partial y}$ can be obtained in a similar fashion. All local transformations of the group can be written as

$$s(f,\alpha) = f + \theta(y\frac{\partial f}{\partial x} + (-x)\frac{\partial f}{\partial y}) - a\frac{\partial f}{\partial x} - b\frac{\partial f}{\partial y} + o(\|\alpha\|^2)(f) \tag{12.49}$$

which corresponds to a linear combination of the 3 basic operators $L_\theta$, $L_a$ and $L_b$[5]. The property which is most important to us is that the 3 operators generate the whole space of local transformations. The result of applying the operators to a function $f$, such as a 2D image for example, is the set of vectors which we have been calling "tangent vector" in the previous sections. Each point in the tangent space correspond to a unique transformation and conversely any transformation of the Lie group (in the example all rotations of any angle and center together with all translations) corresponds to a point in the tangent plane.

[5] These operators are said to generate a Lie algebra, because on top of the addition and multiplication by a scalar, there is a special multiplication called "Lie bracket" defined by $[L_1, L_2] = L_1 \circ L_2 - L_2 \circ L_1$. In the above example we have $[L_\theta, L_a] = L_b$, $[L_a, L_b] = 0$, and $[L_b, L_\theta] = L_a$.

### 12.4.2 Tangent vectors

The last problem which remains to be solved is the problem of coding. Computer images, for instance, are coded as a finite set of discrete (even binary) values. These are hardly the differentiable mappings of $\mathcal{I}$ to $\Re$ which we have been assuming in the previous subsection.

To solve this problem we introduce a smooth interpolating function $C$ which maps the discrete vectors to a continuous mapping of $\mathcal{I}$ to $\Re$. For example, if $P$ is a image of $n$ pixels, it can be mapped to a continuously valued function $f$ over $\Re^2$ by convolving it with a two dimensional Gaussian function $g_\sigma$ of standard deviation $\sigma$. This is because $g_\sigma$ is a differentiable mapping of $\Re^2$ to $\Re$, and $P$ can be interpreted as a sum of impulse functions. In the two dimensional case we can write the new interpretation of $P$ as:

$$P'(x,y) = \sum_{i,j} P[i][j]\delta(x-i)\delta(y-j) \tag{12.50}$$

where $P[i][j]$ denotes the finite vector of discrete values, as stored in a computer. The result of the convolution is of course differentiable because it is a sum of Gaussian functions. The Gaussian mapping is given by:

$$C_\sigma : P \longmapsto f = P' * g_\sigma. \tag{12.51}$$

In the two dimensional case, the function $f$ can be written as:

$$f(x,y) = \sum_{i,j} P[i][j] g_\sigma(x-i, y-j). \tag{12.52}$$

Other coding functions $C$ can be used, such as cubic spline or even bilinear interpolation. Bilinear interpolation between the pixels yields a function $f$ which is differentiable almost everywhere. The fact that the derivatives have two values at the integer locations (because the bilinear interpolation is different on both side of each pixels) is not a problem in practice – just choose one of the two values.

The Gaussian mapping is preferred for two reasons: First, the smoothing parameter $\sigma$ can be used to control the locality of the invariance. This is because when $f$ is smoother, the local approximation of equation (12.45) is valid for larger transformations. And second, when combined with the transformation operator $L$, the derivative can be applied on the closed form of the Gaussian function. For instance, if the X-translation operator $L = \frac{\partial}{\partial x}$ is applied to $f = P' * g_\sigma$, the actual computation becomes:

$$L_X(f) = \frac{\partial}{\partial x}(P' * g_\sigma) = P' * \frac{\partial g_\sigma}{\partial x}. \tag{12.53}$$

because of the differentiation properties of convolution when the support is compact. This is easily done by convolving the original image with the X-derivative of

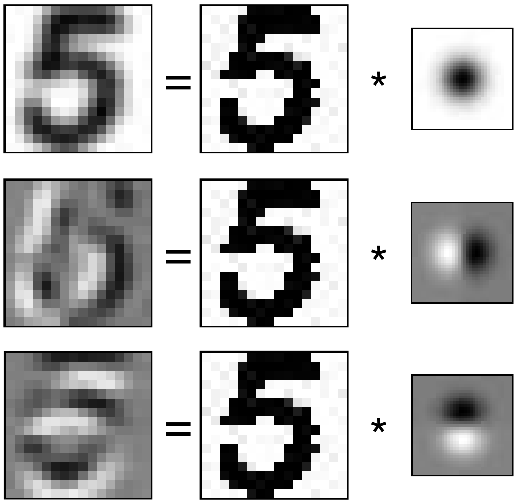

**Fig. 12.12.** Graphic illustration of the computation of $f$ and two tangent vectors corresponding to $L_x = \partial/\partial x$ (X-translation) and $L_x = \partial/\partial y$ (Y-translation), from a binary image $I$. The Gaussian function $g(x, y) = \exp(-\frac{x^2+y^2}{2\sigma^2})$ has a standard deviation of $\sigma = 0.9$ in this example although its graphic representation (small images on the right) have been rescaled for clarity.

the Gaussian function $g_\sigma$. This operation is illustrated in Figure 12.12. Similarly, the tangent vector for scaling can be computed with

$$L_S(f) = \left(x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}\right)(I * g_\sigma) = x(I * \frac{\partial g_\sigma}{\partial x}) + y(I * \frac{\partial g_\sigma}{\partial y}). \tag{12.54}$$

This operation is illustrated in Figure 12.13.

### 12.4.3 Important transformations in image processing

This section summarizes how to compute the tangent vectors for image processing (in 2D). Each discrete image $I_i$ is convolved with a Gaussian of standard deviation $g_\sigma$ to obtain a representation of the continuous image $f_i$, according to equation:

$$f_i = I_i * g_\sigma. \tag{12.55}$$

The resulting image $f_i$ will be used in all the computations requiring $I_i$ (except for computing the tangent vector). For each image $I_i$, the tangent vectors are computed by applying the operators corresponding to the transformations of interest to the expression $I_i * g_\sigma$. The result, which can be precomputed, is an image which is the tangent vector. The following list contains some of the most useful tangent vectors:

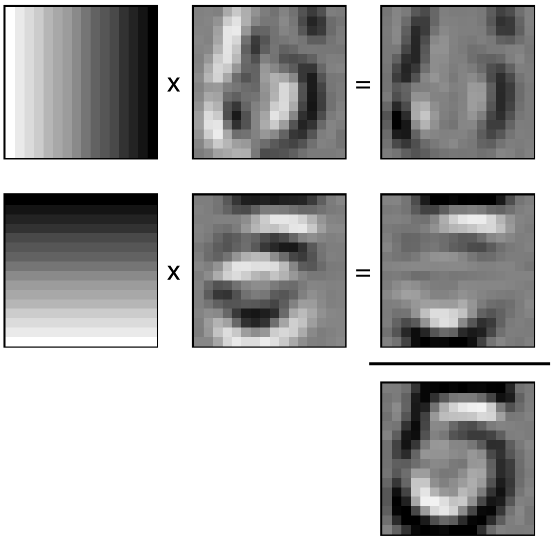

**Fig. 12.13.** Graphic illustration of the computation of the tangent vector $T_u = D_x S_x + D_y S_y$ (bottom image). In this example the displacement for each pixel is proportional to the distance of the pixel to the center of the image ($D_x(x,y) = x - x_0$ and $D_y(x,y) = y - y_0$). The two multiplications (horizontal lines) as well as the addition (vertical right column) are done pixel by pixel.

**X-translation:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha \\ y \end{pmatrix}. \tag{12.56}$$

The Lie operator is defined by:

$$L_X = \frac{\partial}{\partial x}. \tag{12.57}$$

**Y-translation:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x \\ y + \alpha \end{pmatrix}. \tag{12.58}$$

The Lie operator is defined by:

$$L_Y = \frac{\partial}{\partial y}. \tag{12.59}$$

**Rotation:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x\cos\alpha - y\sin\alpha \\ x\sin\alpha + y\cos\alpha \end{pmatrix}. \tag{12.60}$$

The Lie operator is defined by:

$$L_R = y\frac{\partial}{\partial x} + (-x)\frac{\partial}{\partial y}. \tag{12.61}$$

**Scaling:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha x \\ y + \alpha y \end{pmatrix}. \tag{12.62}$$

The Lie operator is defined by:

$$L_S = x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}. \tag{12.63}$$

**Parallel hyperbolic transformation:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha x \\ y - \alpha y \end{pmatrix}. \tag{12.64}$$

The Lie operator is defined by:

$$L_S = x\frac{\partial}{\partial x} - y\frac{\partial}{\partial y}. \tag{12.65}$$

**Diagonal hyperbolic transformation:** This transformation is useful when the classification function is known to be invariant with respect to the input transformation:

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha y \\ y + \alpha x \end{pmatrix}. \tag{12.66}$$

The Lie operator is defined by:

$$L_S = y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}. \tag{12.67}$$

The resulting tangent vector is is the norm of the gradient of the image, which is very easy to compute.

**Thickening:** This transformation is useful when the classification function is known to be invariant with respect to variation of thickness. This is known in morphology as dilation and its inverse, erosion. It is very useful in certain domains (such as handwritten character recognition because) thickening and thinning are natural variations which correspond to the pressure applied on a pen, or to different absorbtion properties of the ink on the paper. A dilation (resp. erosion) can be defined as the operation of replacing each value f(x,y) by the largest (resp. smallest) value of $f(x', y')$ found within a neighborhood of a certain shape, centered at $(x, y)$. The region is called the structural element. We will assume that the structural element is a sphere of radius $\alpha$. We define the thickening transformation as the function which takes the function $f$ and generates the function $f'_\alpha$ defined by:

$$f'_\alpha(X) = \max_{\|r\| \leq \alpha} f(X + r) \quad \text{for } \alpha \geq 0 \tag{12.68}$$

$$f'_\alpha(X) = \min_{\|r\| \leq -\alpha} f(X + r) \quad \text{for } \alpha \leq 0. \tag{12.69}$$

The derivative of the thickening for $\alpha \geq 0$ can be written as:

$$\lim_{\alpha \longrightarrow 0} \frac{f'(X) - f(X)}{\alpha} = \lim_{\alpha \longrightarrow 0} \frac{\max_{\|r\| \leq \alpha} f(X + r) - f(X)}{\alpha}. \tag{12.70}$$

$f(X)$ can be put within the max expression because it does not depend on $\|r\|$. Since $\|\alpha\|$ tends toward 0, we can write:

$$f(X + r) - f(X) = r \cdot \nabla f(X) + O(\|r\|^2) \approx r \cdot \nabla f(X). \tag{12.71}$$

The maximum of

$$\max_{\|r\| \leq \alpha} f(X + r) - f(X) = \max_{\|r\| \leq \alpha} r \cdot \nabla f(X) \tag{12.72}$$

is attained when $r$ and $\nabla f(X)$ are co-linear, that is when

$$r = \alpha \frac{\nabla f(X)}{\|\nabla f(X)\|} \tag{12.73}$$

assuming $\alpha \geq 0$. It can easily be shown that this equation holds when $\alpha$ is negative, because we then try to minimize equation (12.69). We therefore have:

$$\lim_{\alpha \longrightarrow 0} \frac{f'_\alpha(X) - f(X)}{\alpha} = \|\nabla f(X)\| \tag{12.74}$$

which is the tangent vector of interest. Note that this is true for $\alpha$ positive or negative. The same tangent vector describes both thickening and thinning. Alternatively, we can use our computation of the displacement $r$ and define the following transformation of the input:

$$t_\alpha(f) : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha r_x \\ y + \alpha r_y \end{pmatrix} \tag{12.75}$$

where

$$(r_x, r_y) = r = \alpha \frac{\nabla f(X)}{\|\nabla f(X)\|}. \tag{12.76}$$

This transformation of the input space is different for each pattern $f$ (we do not have a Lie group of transformations, but the field structure generated by the (pseudo Lie) operator is still useful. The operator used to find the tangent vector is defined by:

$$L_T = \|\nabla\| \tag{12.77}$$

which means that the tangent vector image is obtained by computing the normalized gray level gradient of the image at each point (the gradient at each point is normalized).

The last 5 transformations are depicted in Figure 12.14 with the tangent vector. The last operator corresponds to a thickening or thinning of the image. This unusual transformation is extremely useful for handwritten character recognition.

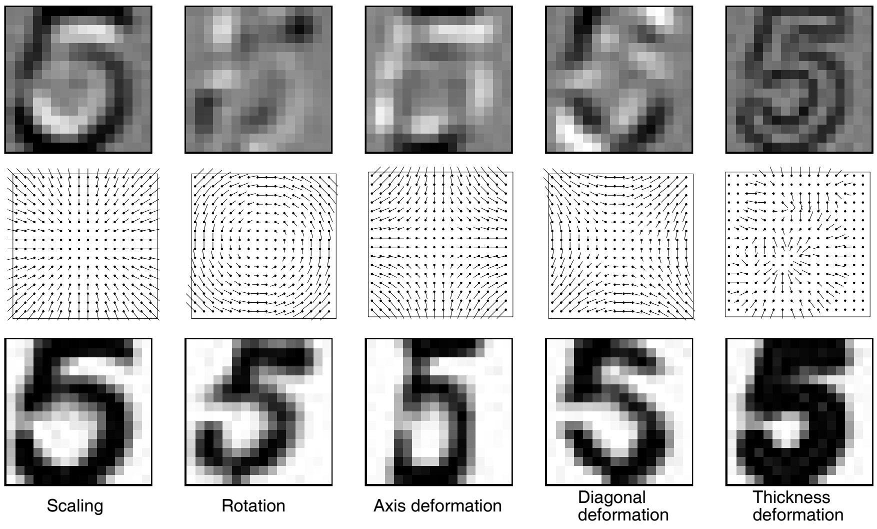

**Fig. 12.14.** Illustration of 5 tangent vectors (top), with corresponding displacements (middle) and transformation effects (bottom). The displacement $D_x$ and $D_y$ are represented in the form of vector field. It can be noted that the tangent vector for the thickness deformation (right column) correspond to the norm of the gradient of the gray level image.

## 12.5 Conclusion

The basic tangent distance algorithm is quite easy to understand and implement. Even though hardly any preprocessing or learning is required, the performance is surprisingly good and compares well to the best competing algorithms. We believe that the main reason for this success is its ability to incorporate *a priori* knowledge into the distance measure. The only algorithm which performed better than tangent distance on one of the three databases was boosting, which has similar *a priori* knowledge about transformations built into it.

Many improvements are of course possible. For instance, smart preprocessing can allow us to measure the tangent distance in a more appropriate "feature" space, instead of the original pixel space. In image classification, for example, the features could be horizontal and vertical edges. This would most likely further improve the performance[6] The only requirement is that the preprocessing must be differentiable, so that the tangent vectors can be computed (propagated) into the feature space.

It is also straightforward to modify more complex algorithms such as LVQ (learning vector quantization) to use a tangent distance. In this case even the tangent vectors can be trained. The derivation has been done for batch training [13] and for on-line training [23] of the tangent vectors. When such training is performed, the *a priori* knowledge comes from other constraints imposed on the tangent vectors (for instance how many tangent vectors are allowed, which classes of transformation do they represent, etc).

Finally, many optimizations which are commonly used in distance based algorithms can be used as successfully with tangent distance to speed up computation. The multi-resolution approach have already been tried successfully [25]. Other methods like "multi-edit-condensing" [1, 30] and K-d tree [4] are also possible.

The main advantage of tangent distance is that it is a modification of a standard distance measure to allow it to incorporate *a priori* knowledge that is specific to the problem at hand. Any algorithms based on a common distance measure (as it is often the case in classification, vector quantization, predictions, etc...) can potentially benefit from a more problem-specific distance. Many of these "distance based" algorithms do not require any learning, which means that they can be adapted instantly by just adding new patterns in the database. These additions are leveraged by the *a priori* knowledge put in the tangent distance.

The two drawbacks of tangent distance are its memory and computational requirements. The most computationally and memory efficient algorithms generally involve learning [20]. Fortunately, the concept of tangent vectors can also be used in learning. This is the basis for the tangent propagation algorithm. The concept is quite simple: instead of learning a classification function from examples of its values, one can also use information about its derivatives. This

[6] There may be an additional cost for computing the tangent vectors in the feature space if the feature space is very complex.

information is provided by the tangent vectors. Unfortunately, not many experiments have been done in this direction. The two main problems with tangent propagation are that the capacity of the learning machine has to be adjusted to incorporate the additional information pertinent to the tangent vectors, and that training time must be increased. After training, the classification time and complexity are unchanged, but the classifier's performance is improved.

To a first approximation, using tangent distance or tangent propagation is like having a much larger database. If the database was plenty large to begin with, tangent distance or tangent propagation would not improve the performance. To a better approximation, tangent vectors are like using a distortion model to magnify the size of the training set. In many cases, using tangent vectors will be preferable to collecting (and labeling!) vastly more training data, and preferable (especially for memory-based classifiers) to dealing with all the data generated by the distortion model. Tangent vectors provide a compact and powerful representation of *a priori* knowledge which can easily be integrated in the most popular algorithms.

**Acknowledgement** P.S. and Y.L. gratefully acknowledge NSF grant No. INT-9726745.

## References

1. P. A.Devijver and J. Kittler. *Pattern Recognition, A Statistical Approache.* Prentice Hall, Englewood Cliffs, 1982.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structure and Algorithms.* Addison-Wesley, 1983.
3. L. Bottou and V. N. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.
4. A. J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recognition*, 23:171–178, 1990.
5. D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.
6. Y. Choquet-Bruhat, C. DeWitt-Morette, and M. Dillard-Bleick. *Analysis, Manifolds and Physics.* North-Holland, Amsterdam, Oxford, New York, Tokyo, 1982.
7. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
8. B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern classification Techniques.* IEEE Computer Society Press, Los Alamitos, California, 1991.
9. H. Drucker, R. Schapire, and P. Y. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7, No. 4:705–719, 1993.
10. K. Fukunaga and T. E. Flick. An optimal global nearest neighbor metric. *IEEE transactions on Pattern analysis and Machine Intelligence*, 6, No. 3:314–318, 1984.
11. R. Gilmore. *Lie Groups, Lie Algebras and some of their Applications.* Wiley, New York, 1974.
12. T. Hastie, E. Kishon, M. Clark, and J. Fan. A model for signature verification. Technical Report 11214-910715-07TM, AT&T Bell Laboratories, July 1991.

13. T. Hastie and P. Y. Simard. Metrics and models for handwritten character recognition. *Statistical Science*, 13, 1998.
14. T. J. Hastie and R. J. Tibshirani. *Generalized Linear Models.* Chapman and Hall, London, 1990.
15. G. E. Hinton, C. K. I. Williams, and M. D. Revow. Adaptive elastic models for hand-printed character recognition. In *Advances in Neural Information Processing Systems*, pages 512–519. Morgan Kaufmann Publishers, 1992.
16. A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for non-orthogonal problems. *Technometrics*, 12:55–67, 1970.
17. T. Kohonen. Self-organization and associative memory. In *Springer Series in Information Sciences*, volume 8. Springer-Verlag, 1984.
18. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, (Denver, 1989), 1990. Morgan Kaufman.
19. Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.
20. Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition. In J. H. Oh, C. Kwon, and S. Cho, editors, *Neural Networks: The Statistical Mechanics Perspective*, pages 261–276. World Scientific, 1995.
21. E. Parzen. On estimation of a probability density function and mode. *Ann. Math. Stat.*, 33:1065–1076, 1962.
22. W. H. Press, B. P. Flannery, Teukolsky S. A., and Vetterling W. T. *Numerical Recipes.* Cambridge University Press, Cambridge, 1988.
23. H. Schwenk. The diabolo classifier. *Neural Computation*, in press, 1998.
24. R. Sibson. Studies in the robustness of multidimensional scaling: Procrustes statistices. *J. R. Statist. Soc.*, 40:234–238, 1978.
25. P. Y. Simard. Efficient computation of complex distance metrics using hierarchical filtering. In *Advances in Neural Information Processing Systems.* Morgan Kaufmann Publishers, 1994.
26. F. Sinden and G. Wilfong. On-line recognition of handwritten symbols. Technical Report 11228-910930-02IM, AT&T Bell Laboratories, June 1992.
27. V. N. Vapnik. *Estimation of dependences based on empirical data.* Springer Verlag, 1982.
28. V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Th. Prob. and its Applications*, 17(2):264–280, 1971.
29. N. Vasconcelos and A. Lippman. Multiresolution tangent distance for affine-invariant classification. In *Advances in Neural Information Processing Systems*, volume 10, pages 843–849. Morgan Kaufmann Publishers, 1998.
30. J. Voisin and P. Devijver. An application of the multiedit-condensing technique to the reference selection problem in a print recognition system. *Pattern Recogntion*, 20 No 5:465–474, 1987.