Transformation Invariance in Pattern Recognition: Tangent Distance and Propagation

Patrice Y. Simard,[1] Yann A. Le Cun,[2] John S. Denker,[2] Bernard Victorri[3]

[1] Microsoft Research, 1 Microsoft Way, Redmond, WA 98052. E-mail: patrice@microsoft.com

[2] AT&T Labs, 100 Schulz Dr., Redbank, NJ, 07701. E-mail: (Y.A.L): yann@research.att.com; E-mail (J.S.D.): jsd@research.att.com

[3] LATTICE-CNRS, ENS Paris, France. E-mail: victorri@ens.fr

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. 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. Int J Imaging Syst Technol, 11, 181–197, 2000

I. 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 and objects in different categories may have quite similar features. A typical example is handwritten digit recognition. Characters, typically represented as fixed-size images (e.g., 16×16 pixels), must be classified into 1 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 and 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. This is because memory-based algorithms often rely on a metric that can be modified to incorporate transformation invariances; the 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.

A. 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 with all the stored prototypes. 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 (Parzen, 1962) or learned (Broomhead and Lowe, 1988) coefficients. Storage schemes vary from storing the entire training set and picking appropriate subsets of it (Dasarathy, 1991) to storing learned patterns such as learning vector quantization (LVQ; Kohonen, 1984). 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 (Fukunaga and Flick, 1984) or in elastic matching methods (Hinton et al., 1992).

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 large set of proto-

Correspondence to: P. Simard
The majority of this work was completed at AT&T Labs.

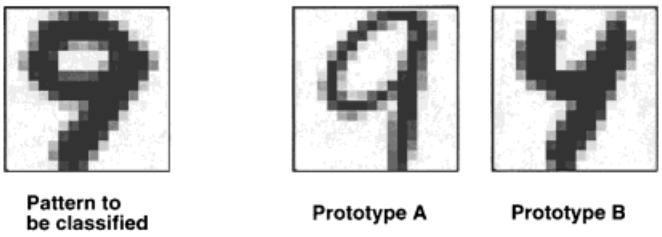

Figure 1. According to the Euclidean distance, the pattern to be classified is more similar to prototype B. A better distance measure would find that prototype A is closer because it differs mainly by a rotation and a thickness transformation, two transformations that should leave the classification invariant.

types. 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, and skews must be stored. In real situations, this approach leads to impractically large prototype sets or to mediocre recognition accuracy (Fig. 1). An unlabeled image of a thick, slanted "9" must be classified by finding the closest prototype image from 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 use a so-called feature extractor. Its 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 1 can be solved by deslanting and thinning the images.

An alternative 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 (Hinton et al., 1992). With the example of Figure 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 16×16 pixel character image can be viewed as the components of a 256-dimensional (256-D) vector. One pattern, or one prototype, is a point in this 256-D 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 $s(P, \alpha)$ that depends on one parameter α (e.g., the angle of the rotation), the set of all the transformed patterns

$$S_P = \{x | \exists\, \vec{\alpha} \text{ for which } x = s(P, \alpha)\} \tag{1}$$

is a 1-D curve in the vector space of the inputs. In the remainder of this study, we will always assume that we have chosen s to be differentiable with respect to both P and α, such that $s(P, 0) = P$.

When the set of transformations is parameterized by n parameters α_i, the intrinsic dimension of the manifold S_P is n. For example, if the allowable transformations of character images are horizontal and vertical shifts, rotations, and scaling, the surface will be a 4-D manifold.

In general, the manifold will not be linear. Even a simple image translation corresponds to a highly nonlinear 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 nonlinearity 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 and is sometimes referred to as Procrustes analysis (Sibson, 1978). It has been applied to signature verification (Hastie et al., 1991) and on-line character recognition (Sinden and Wilfong, 1992).

This study considers the more general case of nonlinear transformations such as geometric transformations of gray-level images. Remember that even a simple image translation corresponds to a highly nonlinear transformation in the high-dimensional pixel space. The main idea of this study 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 2, in the case of rotation for handwritten digits. The theoretical curve in pixel space which represents Eq. (1), together with its linear approximation, is shown in Figure 2 (top). Points of the transformation curve are depicted below for various amounts of rotation (each angle corresponds to a value of α). Figure 2 (bottom) depicts the linear approximation of the curve $s(P, \alpha)$ given by the Taylor expansion of s around $\alpha = 0$:

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

This linear approximation is completely characterized by the point P and the tangent vector $T = \dfrac{\partial s(P, \alpha)}{\partial \alpha}$. Tangent vectors, also called the Lie derivatives of the transformation s, will be discussed in

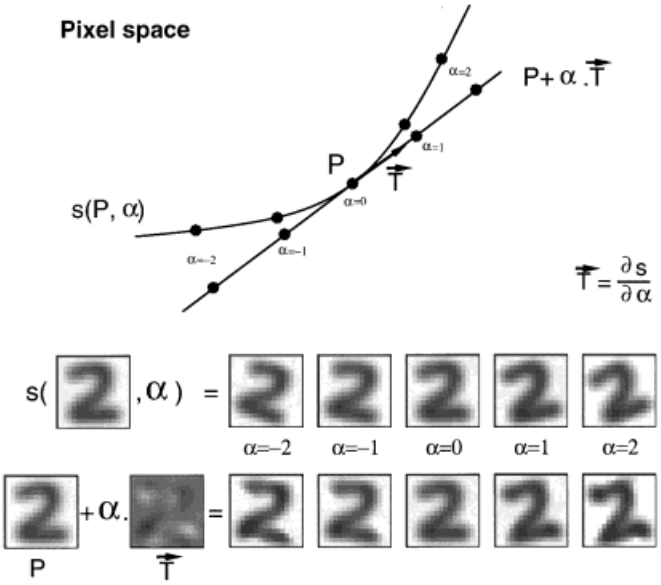

Figure 2. (Top) Representation of the effect of the rotation in pixel space. (Middle) Small rotations of an original digitized image of the digit "2" for different angle values of α. (Bottom) Images obtained by moving along the tangent to the transformation curve for the same original digitized image *P* by adding various amounts (α) of the tangent vector *T*.

Section IV. For reasonably small angles ($\|\alpha\| < 1$), the approximation is very good (Fig. 2).

Figure 3 illustrates the difference among the Euclidean distance, the full invariant distance (minimum distance between manifolds), and the tangent distance. Both the prototype and the pattern are deformable (two-sided distance). However, for simplicity or efficiency reasons, it is also possible to deform only the prototype or

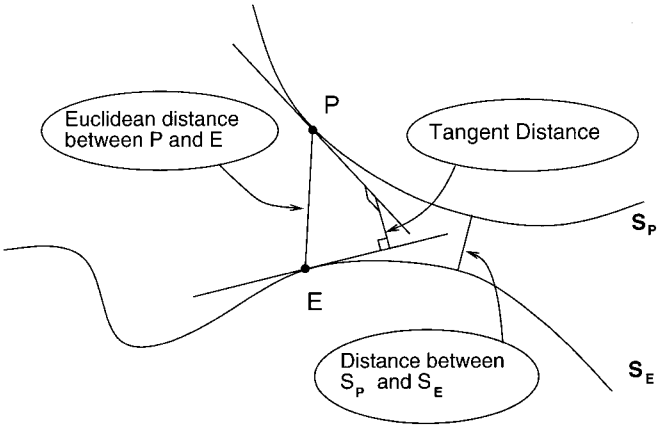

Figure 3. Illustration of the Euclidean distance and the tangent distance between *P* and *E*. The curves S_p and S_e represent the sets of points obtained by applying the chosen transformations (e.g., translations and rotations) to *P* and *E*. The lines going through *P* and *E* represent the tangent 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.

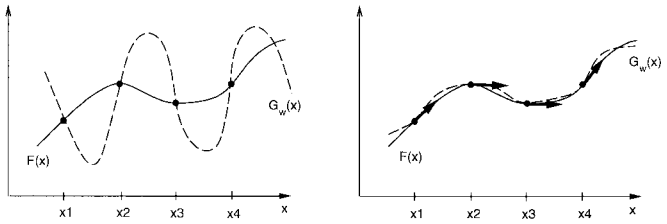

Figure 4. Learning a given function (solid line) from a limited set of examples (x_1 to x_4). The fitted curves are shown by a dotted line. (Left) The only constraint is that the fitted curve goes through the examples. (Right) The fitted curves go through each example and its derivatives evaluated at the examples agree with the derivatives of the given function.

only the unknown pattern (one-sided distance). Although we concentrate on using tangent distance to recognize images, the method can be applied to many different types of signals, such as temporal signals, speech, and sensor data.

B. 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, and regression.

We assume that all data are 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 ws (Fig. 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 Figure 4, 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 of 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.

Selecting a good family $\mathcal{G} = \{G_w, w \in \mathcal{R}^q\}$ of functions is a difficult task, sometimes known as model selection (Hastie and Tibshirani, 1990; Hoerl and Kennard, 1970). 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). However, 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 that 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 (Vapnik, 1982; Vapnik and Chervonenkis, 1971). 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 are 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 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. Because 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 the derivatives of G_w 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 (Fig. 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, and scaling. In other words, the directional derivatives of the classification function in the directions of these transformations is zero (Fig. 4).

The right part of Figure 4 shows how the additional constraints on G_w help generalization by constraining the values of G_w outside the training set. For every transformation that 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 Fig. 2), computed from the curve of transformation.

Section II analyzes in detail how to use distance based on a tangent vector in memory-based algorithms; Section III discusses the use of tangent vectors in neural networks, with the tangent propagation algorithm; and Section IV compares different algorithms to compute tangent vectors.

II. 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. Therefore, it is invariant with respect to the transformation used to generate S_P and S_E (Fig. 3). Unfortunately, these manifolds have no analytic expression in general. Finding the distance between them is a difficult optimization problem with multiple local minima. Besides, true invariance is not necessarily desirable because 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 nonlinear manifolds S_P and S_E. This solves three problems at once: (1) linear manifolds have simple analytic expressions that can be easily computed and stored, (2) finding the minimum distance between linear manifolds is a simple least-squares problem that can be solved efficiently, and (3) this distance is locally, 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 3.

Figure 3 represents two patterns P and E in 3-D space. The manifolds generated by s are represented by 1-D curves going through E and P, respectively. The linear approximations to the manifolds are represented by lines tangential to the curves at E and P. These lines do not intersect in three dimensions and the shortest distance between them (uniquely defined) is $D(E, P)$. The distance between the two nonlinear transformation curves $\mathcal{D}(E, P)$ is also shown on Figure 3.

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

A. Implementation. 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 α. We require s to be differentiable with respect to α and P and require $s(P, 0) = P$. For example, if P is a 2-D image, $s(P, \alpha)$ could be a rotation of P by the angle α. If we are interested in all transformations of images that conserve distances (isometry), $s(P, \alpha)$ would be a rotation by α_θ followed by a translation by α_x, α_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.

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

$$L_P = \left.\frac{\partial s(P, \alpha)}{\partial \alpha}\right|_{\alpha=\vec{0}} = \left[\frac{\partial s(P, \alpha)}{\partial \alpha_1}, \ldots, \frac{\partial s(P, \alpha)}{\partial \alpha_m}\right]_{\alpha=\vec{0}} \quad (3)$$

which are vectors tangential 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:

$$D(E, P) = \min_{x \in T_E, y \in T_P} \|x - y\|^2 \quad (4)$$

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

[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.

$$E'(\alpha_E) = E + L_E\alpha_E \tag{5}$$

$$P'(\alpha_P) = P + L_P\alpha_P \tag{6}$$

where L_E and L_P are the matrices containing the tangent vectors (Eq. 3) and the vectors α_E and α_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 α_E denote vectors and matrices in linear Eq. (5). For example, if the pixel space was of dimension 5, and there were two tangent vectors, we could rewrite Eq. (5) as:

$$\begin{bmatrix} E'_1 \\ E'_2 \\ E'_3 \\ E'_4 \\ E'_5 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \end{bmatrix} + \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \\ L_{31} & L_{32} \\ L_{41} & L_{42} \\ L_{51} & L_{52} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \tag{7}$$

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:

$$D(E, P) = \min_{\alpha_E,\alpha_P} \|E'(\alpha_E) - P'(\alpha_P)\|^2 = \min_{\alpha_E,\alpha_P} d(\alpha_E,\alpha_p) \tag{8}$$

where $d(\alpha_E, \alpha_p) = \|E'(\alpha_E) - P'(\alpha_P)\|_2$, amounts to solving a linear least-squares problem. In the interest of clarity and to make the computational costs more apparent, the details of the computation of α_P and α_E are spelled out (the advanced reader can skip to the next section). The optimality condition is that the partial derivatives of $d(\alpha_E, \alpha_P)$ with respect to α_P and α_E should be zero:

$$\frac{\partial d(\alpha_E, \alpha_P)}{\partial \alpha_E} = 2(E'(\alpha_E) - P'(\alpha_P))^\top L_E = 0 \tag{9}$$

$$\frac{\partial d(\alpha_E, \alpha_P)}{\partial \alpha_P} = 2(P'(\alpha_P) - E'(\alpha_E))^\top L_P = 0 \tag{10}$$

Substituting E' and P' by their expressions yields to the following linear system of equations, which we must solve for α_P and α_E:

$$L_P^\top(E - P - L_P\alpha_P + L_E\alpha_E) = 0 \tag{11}$$

$$L_E^\top(E - P - L_P\alpha_P + L_E\alpha_E) = 0 \tag{12}$$

The solution of this system is:

$$(L_{PE}L_{EE}^{-1}L_E^\top - L_P^\top)(E - P) = (L_{PE}L_{EE}^{-1}L_{EP} - L_{PP})\alpha_P \tag{13}$$

$$(L_{EP}L_{PP}^{-1}L_P^\top - L_E^\top)(E - P) = (L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \tag{14}$$

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, α_P and α_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 α_P and α_E in Eqs. (5) and (6). If n is the dimension of the input space (i.e., the length of vectors 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.

B. 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. Figure 5 (bottom) 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 10 curves for Euclidean distance and 10 more curves for tangent distance, measuring the distance between the translated reference and 1 of the 10 digits.

Because 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 Figure 5 that if the reference (the image "3") is translated by more than two 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 five pixels. Similar curves were obtained with all the other transformations (e.g., rotation and scaling).

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. 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 6 further illustrates this phenomenon by displaying points in the tangent plane generated from only five 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 that 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

[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°). Global invariance is not desirable in digit recognition, because we need to distinguish a "6" from a "9."

[3] An isomorphism actually, see "Lie algebra" in (Choquet-Bruhat et al., 1982)

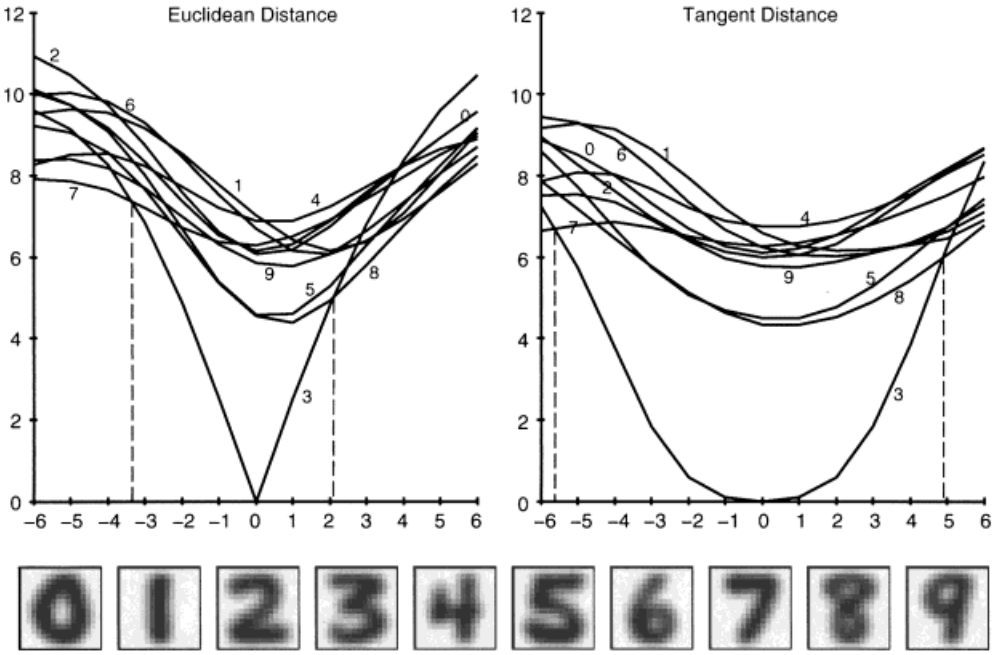

Figure 5. Euclidean and tangent distances among 10 typical images of handwritten digits and a translated image of the digit "3." The abscissa represents the amount of horizontal translation (measured in pixels).

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 nonlinear transforms in pixel space.) Additional transformations have been tried with less success. Three databases were used to test our algorithm:

1. U.S. Postal Service (USPS) database: The database consisted of 16×16 pixel size-normalized images of handwritten digits taken from U.S. mail envelopes. The training and testing set had, respectively, 9,709 and 2,007 examples.
2. 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.
3. NIST2 database: The third experiment was performed on a database made out of the training and testing database provided by the NIST (see above). NIST had divided the data into two sets, which unfortunately had different distributions. The NIST1 training set (223,000 patterns) was easier than the testing set (59,000 patterns). In our NIST2 experiments, we combined these two sets 50/50 to make a training set of 60,000 patterns and testing and 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 the

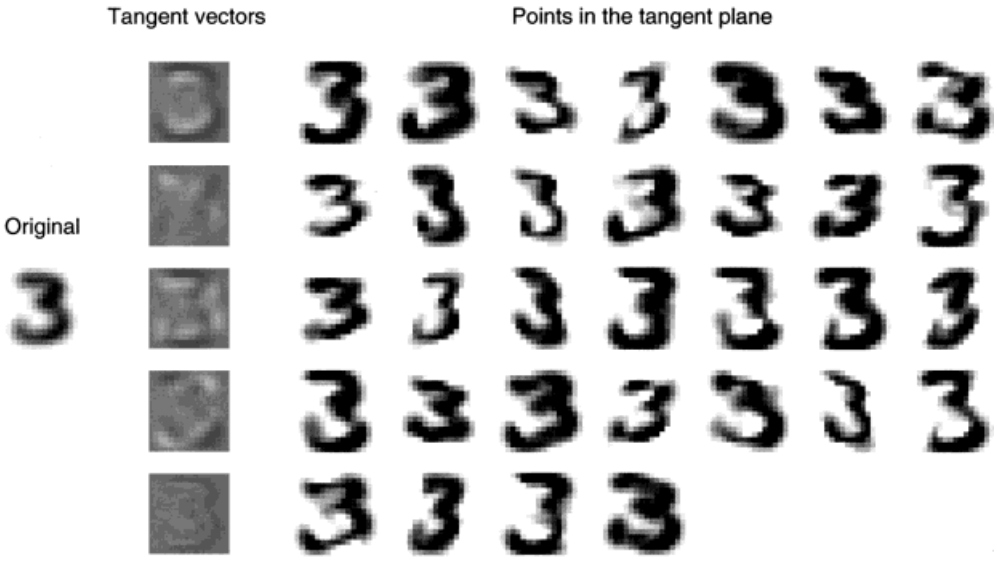

Figure 6. (Left) Original image. (Middle) Five tangent vectors corresponding, respectively, to the five transformations: scaling, rotation, expansion of the X axis while compressing the Y axis, expansion of the first diagonal while compressing the second diagonal, and thickening. (Right) 32 points in the tangent space generated by adding or subtracting each of the five tangent vectors.

Table I. Performances in percent of errors for (in order) human, K-nearest neighbor (K-NN), tangent distance (TD), Lenet1 (simple neural network), Lenet4 (large neural network), optimal margin classifier (OMC), local learning (LL) and boosting (Boost).

	Human	K-NN	TD	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

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 NIST. 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 were 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, whereas 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 seven transformations. For the USPS and NIST2 databases, the prototype set was constructed as before. However, for NIST1, it was constructed by cycling through the training set. Any patterns that 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 (LeCun et al., 1990, 1995), optimal margin classifier (Cortes and Vapnik, 1995), local learning (Bottou and Vapnik, 1992), and boosting (Drucker et al., 1993) were also used on these databases. A case study can be found in LeCun et al. (1995). The results are summarized in Table 1.

As illustrated in Table 1, the 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 compared with local learning, optimal margin classifier, and boosting. The advantage of tangent distance is the a priori knowledge of transformation invariance embedded into the distance. When the training data are 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.

C. 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 (e.g., they are not always theoretically sound, page area is too valuable, the tricks are specific to one particular application, and commercial competitive considerations discourage telling everyone how to reproduce the result). However, 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 α. They can be written as:

$$L_P = \left.\frac{\partial s(P, \alpha)}{\partial \alpha}\right| = \lim_{\epsilon \to 0} \frac{s(P, \epsilon) - s(P, 0)}{\epsilon} \tag{15}$$

It is therefore important that s be differentiable (and well behaved) with respect to α. In particular, it is clear from Eq. (15) that $s(P, \epsilon)$ must be computed for ϵ 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_σ as a preprocessing for P, such that $s(C_\sigma(P), \alpha)$ is differentiable (with respect to $C_\sigma(P)$ and α, 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 σ. If $s(C_\sigma(P), \alpha)$ is a translation of α pixels, the derivative of $s(C_\sigma(P), \alpha)$ can easily be computed. This is because $s(C_\sigma(P), \epsilon)$ can be obtained by translating Gaussian functions. Preprocessing is discussed in more detail in Section IV.

The smoothing factor σ controls the locality of the invariance. The smoother the transformation curve defined by s, the longer the linear approximation will be valid. In general, the best smoothing is the maximum smoothing that does not blur the features. For example, in handwritten character recognition with 16×16 pixel images, a Gaussian function with a standard deviation of one pixel yielded the best results. Increased smoothing led to confusion (such as a "5" mistaken for a "6" because the lower loop had been closed by the smoothing) and decreased smoothing did not 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 Eq. (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 are 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 simple and elegant. Equation (8) can be replaced by Eq. (16):

$$D(E, P) = \min_{\alpha_E, \alpha_P} \|E + L_E\alpha_E - P - L_P\alpha_P\|^2 + k\|L_E\alpha_E\|^2 + k\|L_P\alpha_P\|^2 \tag{16}$$

The physical interpretation of this equation is illustrated in Fig. 7. The point $E'(\alpha_E)$ on the tangent plane T_E is attached to E with a spring with 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 Eq. (16) with respect to α_E and α_P. The differentiation yields:

$$L_P^\top(E - P - L_P(1 + k)\alpha_P + L_E\alpha_E) = 0 \tag{17}$$

$$L_E^\top(E - P - L_P\alpha_P + L_E(1 + k)\alpha_E) = 0 \tag{18}$$

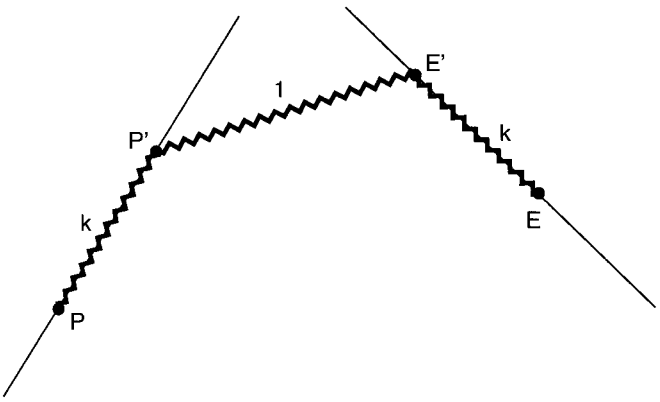

Figure 7. The tangent distance between E and P is the elastic energy stored in each of the three springs connecting P, P', E', and E. P' and E' can move without friction along the tangent planes. The spring constants are indicated on the figure.

The solution of this system is

$$(L_{PE}L_{EE}^{-1}L_E^\top - (1+k)L_P^\top)(E-P) = (L_{PE}L_{EE}^{-1}L_{EP} - (1+k)^2 L_{PP})\alpha_P \tag{19}$$

$$(L_{EP}L_{PP}^{-1}L_P^\top - (1+k)L_E^\top)(E-P) = ((1+k)^2 L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \tag{20}$$

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 $k > 0$ and is more numerically stable. Note that in the limit cases, the system yields the standard tangent distance ($k = 0$) and the Euclidean distance ($k = \infty$). This approach is also 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 the dimension of the input space is large compared with the number of tangent vectors, keeping k as small as possible is better. This is because it does not 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 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 sufficient confidence that a sample is 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 that can greatly reduce the computation of each classification. 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 the tangent distance (for instance, D_1 to D_5 could be the Euclidean distance with increasing resolution and D_6 to D_{10} could each add a tangent vector at full resolution).

The basic idea is to keep a pool of all the prototypes that 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 three steps at each stage and proceeds from Stage 1 to Stage m or until the classification is complete. In Step 1, the distance D_i among all the samples in the pool and the unclassified pattern is computed. In Step 2, a classification and a confidence score is computed with these distances. If the confidence is good enough, that is, better than C_i (e.g., if all the samples left in the pool are in the same class), the classification is complete; otherwise, proceed to Step 3. In Step 3, the K_i closest samples, according to distance D_i, are kept in the pool and 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) because these elements need not to be sorted (Aho et al., 1983; Press et al., 1988). 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 are a minimum K_i and a 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:

$$\begin{aligned}\text{computational cost} \approx \sum_i & \text{ number of prototypes at stage } i \\ & \times \text{ distance complexity at stage } i \\ & \times \text{ probability to reach stage } i\end{aligned} \tag{21}$$

All this is better illustrated in Figure 8. This system was used for the USPS experiment. In classification of handwritten digits (16×16 pixel images), D_1, D_2, and D_3 were the Euclidean distances at resolution 2×2, 4×4, and 8×8 respectively. D_4 was the one-sided tangent distance with X-translation, on the sample side only, at resolution 8×8. D_5 was the double-sided tangent distance with X-translation at resolution 16×16. Each of the subsequent

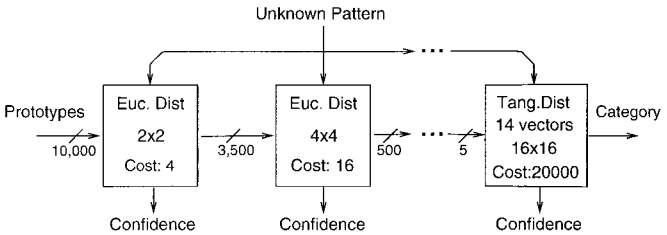

Figure 8. Pattern recognition using a hierarchy of distances. The filter proceeds from left (starting with the whole database) to right (where only a few prototypes remain). At each stage, distances between prototypes and the unknown pattern are computed and sorted; the best candidate prototypes are 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.

Table II. Summary computation for the classification of one pattern.

i	No. T.V.	Resolution	No. Prototypes (K_i)	No. Dot Products	Probability	No. 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

The first column is the distance index; the second column indicates the number of tangent vectors (0 for the Euclidean distance); 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 that must be computed to evaluate distance D_i; the sixth column indicates the probability to not skip that stage after the confidence score has been used; and the last column indicates the total average number of multiply-adds that must be performed (product of Columns 3 to 6) at each stage.

distances added one tangent vector on each side (Y-translation, scaling, rotation, hyperbolic deformation 1, hyperbolic deformation 2, and thickness) until the full tangent distance was computed (D_{11}).

Table 2 shows the expected number of multiply-adds at each stage. 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 six times faster than computing the the Euclidean distance at full resolution).

Multiple Iterations. Tangent distance can be viewed as one iteration of a Newton-type algorithm that finds the points of minimum distance on the true transformation manifolds. The vectors α_E and α_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 (nonlinear) transformations. In other words, α_E and α_P can be used to compute the points $s(E, \alpha_E)$ and $s(P, \alpha_P)$, the real nonlinear 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.

This process did not improve handwritten character recognition, but yielded impressive results in face recognition (Vasconcelos and Lippman, 1998). In that case, each successive iteration was done at increasing resolution (hence, combining hierarchical distances and multiple iterations), making the whole process computationally efficient.

III. 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 classification function by way of optimization of its parameters. More precisely, assume the classification function can be written as $G_w(x)$, where x is the input to the classifier and w is a parameter vector that must be optimized to yield good classification. We present an algorithm, called tangent propagation, in which gradient descent in w is used to improve both classification and transformation invariance of the training data.

In a neural network context, the process can be viewed as a generalization of the widely used back propagation method, which propagates information about the training data. The exception is that the new algorithm also propagates transformation invariance information.

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

$$\varepsilon = \int \|G_w(x) - F(x)\|^2 d\mathcal{P}(x) \tag{22}$$

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

$$E = \sum_{i=1}^{p} \|G_w(x_i) - F(x_i)\|^2 \tag{23}$$

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:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{24}$$

Consider an input transformation $s(x, \alpha)$ controlled by a parameter α. 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 $\dfrac{\partial F(s(x_i, \alpha))}{\partial \alpha}$ 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:

$$E_r = \sum_{i=1}^{p} \left\| \frac{\partial G_w(s(x_i, \alpha))}{\partial \alpha} - \frac{\partial F(s(x_i, \alpha))}{\partial \alpha} \right\|^2_{\alpha=0} \tag{25}$$

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 Eq. (25) to:

$$E_r = \sum_{i=1}^{p} \left\| \frac{\partial G_w(s(x_i, \alpha))}{\partial \alpha} \right\|^2_{\alpha=0} \tag{26}$$

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

$$E = \eta E_p + \mu E_r \tag{27}$$

with the weight update rule:

$$\Delta w = -\frac{\partial E}{\partial w} \tag{28}$$

The learning rates (or regularization parameters) η and μ are tremendously important. This is because they determine the tradeoff between learning the invariances (based on the chosen directional derivatives) vs. 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 Eq. (26), can be written as:

$$\left.\frac{\partial G_w(s(x, \alpha))}{\partial \alpha}\right|_{\alpha=0} = \left.\frac{\partial G_w(s(x, \alpha))}{\partial s(x, \alpha)} \frac{\partial s(x, \alpha)}{\partial \alpha}\right|_{\alpha=0} = \nabla_x G_w(x) \cdot \left.\frac{\partial s(x, \alpha)}{\partial \alpha}\right|_{\alpha=0} \tag{29}$$

because $s(x, \alpha) = x$ if $\alpha = 0$. 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 α is multidimensional, the forward propagation must be repeated for each tangent vector.

The theory of Lie algebras (Gilmore, 1974) ensures that compositions of local (small) transformations correspond to linear combinations of the corresponding tangent vectors (this result is discussed further in Section IV). 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 horizontal 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 (Eq. 28). It is analogous to ordinary back propagation. In addition to propagating neuron activations, it also propagates the tangent vectors. The equations can be easily derived from Figure 9.

A. Local Rule. The forward propagation equation is:

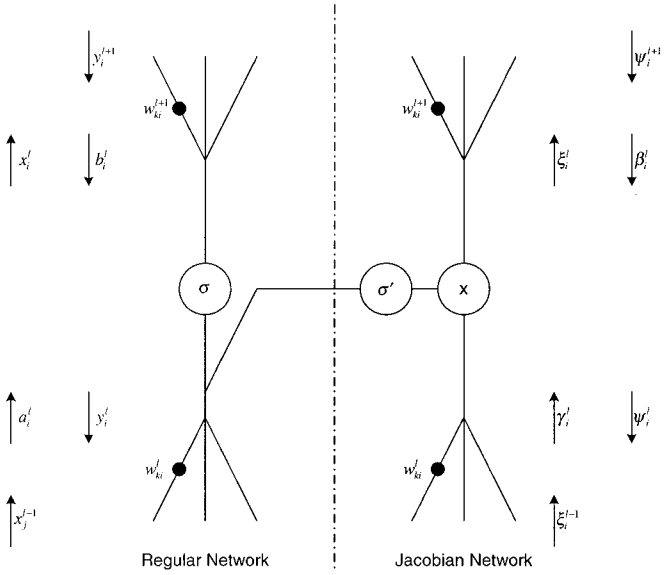

Figure 9. Forward (a, x, γ, ξ) and backward propagated variables (b, y, β, ψ) in the regular (roman symbols) and the Jacobian (linearized) network (Greek symbols). Converging forks (in the direction in which the signal is traveling) are sums; diverging forks duplicate the values.

$$a_i^l = \sum_j w_{ij}^l x_j^{l-1} \quad x_i^l = \sigma(a_i^l) \tag{30}$$

Where σ is a nonlinear 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:

$$\gamma_i^l = \sum_j w_{ij}^l \xi_j^{l-1} \quad \xi_i^l = \sigma'(a_i^l)\gamma_i^l \tag{31}$$

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

$$\frac{\partial E}{\partial \xi_i^l} = \sum_k \frac{\partial E}{\partial \gamma_k^{l+1}} \frac{\partial \gamma_k^{l+1}}{\partial \xi_i^l} \qquad \frac{\partial E}{\partial \gamma_i^l} = \frac{\partial E}{\partial \xi_i^l} \frac{\partial \xi_i^l}{\partial \gamma_i^l} \tag{32}$$

$$\beta_i^l = \sum_k \psi_k^{l+1} w_{ki}^{l+1} \qquad \psi_i^l = \beta_i^l \sigma'(a_i^l) \tag{33}$$

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

$$\frac{\partial E}{\partial x_i^l} = \sum_k \frac{\partial E}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial x_i^l} \qquad \frac{\partial E}{\partial a_i^l} = \frac{\partial E}{\partial x_i^l} \frac{\partial x_i^l}{\partial a_i^l} + \frac{\partial E}{\partial \xi_i^l} \frac{\partial \xi_i^l}{\partial a_i^l} \tag{34}$$

$$b_i^l = \sum_k y_k^{l+1} w_{ki}^{l+1} \qquad y_i^l = b_i^l \sigma'(a_i^l) + \beta i \sigma''(a_i^l)\gamma_i^l \tag{35}$$

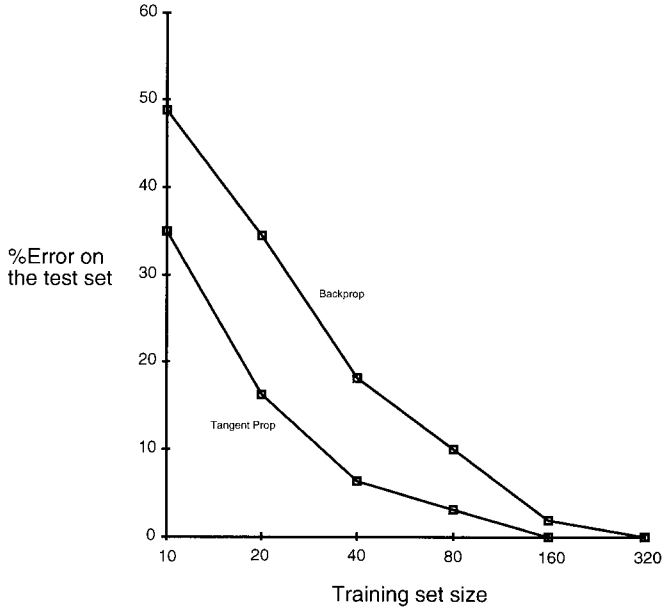

Figure 10. Generalization performance curve as a function of the training set size for the tangent and back prop algorithms.

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:

$$\Delta w_{ij}^l = -\frac{\partial E}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_{ij}^l} - \frac{\partial E}{\partial \gamma_i^l}\frac{\partial \gamma_i^l}{\partial w_{ij}^l} \tag{36}$$

$$\Delta w_{ij}^l = -y_i^l x_j^{l-1} - \psi_i^l \xi_j^{l-1} \tag{37}$$

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 that is not trained for invariance (except that the weights have different values).

B. Results. Two experiments illustrate the advantages of tangent prop. The first experiment is a classification task, using a small (linearly separable) set of 480 binarized 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 (5,194 connections, 1,060 free parameters; LeCun, 1989). The generalization performance as a function of the training set size for traditional back and tangent prop is compared in Figure 10. We conducted additional experiments in which we implemented translations, rotations, expansions, and hyperbolic deformations. This set of six generators is a basis for all linear transformations of coordinates for 2-D 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 are highly correlated, tangent prop yields a large speed advantage. Because the distortion model implies adding lots of highly correlated data, the advantage of tangent prop over the distortion model becomes clear.

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 (Fig. 11, bottom). The network has 1 input unit, 20 hidden units, and 1 output unit. Two strategies are possible: either generate a small set of training points covering each of the plateaus (open squares on Fig. 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 11 (top). Each sweep through the training set for tangent prop is a little faster because it requires only six forward propagations, whereas it requires nine in the distortion model. As can be seen, stable performance is achieved after 1,300 sweeps for the tangent prop, vs. 8,000 for the distortion model. The overall speedup is therefore about 10.

In this example, tangent prop can take advantage of a large regularization term. The distortion model is at a disadvantage because the only parameter that effectively controls the amount of

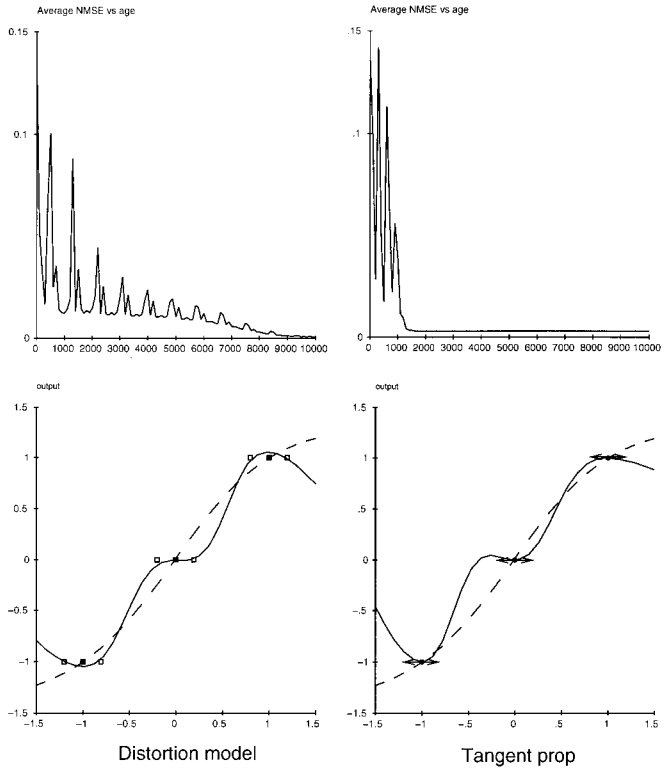

Figure 11. Comparison of the distortion model (left) and tangent prop (right). The top row gives the learning curves (error vs. number of sweeps through the training set). The bottom row gives the final input-output function of the network; the dashed line is the result for unadorned back prop.

regularization is the magnitude of the distortions. This cannot be increased to large values because the right answer is only invariant under small distortions.

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 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. Because the tangent vectors introduce even more correlation inside the training set, substantial speedup 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 three tangent vectors, the training sequence could be:

$$x_1,\ t_1(x_1),\ x_2,\ t_2(x_2),\ x_3,\ t_3(x_3),\ x_4,\ t_1(x_4),\ x_5,\ t_2(x_5),\ \ldots \quad (38)$$

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.

IV. TANGENT VECTORS

We consider the general paradigm for transformation invariance and for the tangent vectors used in the two previous sections. Before we introduce each transformation and its corresponding tangent vectors, the theory behind the practice is explained. 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 and rotation of $\mathcal{R}^2$) and their effect on a functional of that space (such as a mapping of $\mathcal{R}^2$ to $\mathcal{R}$, which may represent an image, in continuous form). The theory of Lie groups and Lie algebra (Choquet-Bruhat et al., 1982) allows us to do this. The second problem involves coding. Computer images are finite vectors of discrete variables. How can a theory that was developed for differentiable functional of $\mathcal{R}^2$ to $\mathcal{R}$ be applied to these vectors?

We provide a brief explanation of the theorems of Lie groups and Lie algebras, which are applicable to pattern recognition. We also explore solutions to the coding problem. Finally, some examples of transformation and coding are given for particular applications.

A. Lie Groups and Lie Algebras. Consider an input space $\mathcal{I}$ (e.g., the plane $\mathcal{R}^2$) and a differentiable function f that maps points of $\mathcal{I}$ to $\mathcal{R}$.

$$f\colon X \in B \to f(X) \in \mathcal{R} \quad (39)$$

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

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

$$t_\alpha\colon X \in \mathcal{I} \to t_\alpha(X) \in \mathcal{I} \quad (40)$$

We assume that t_α is differentiable with respect to α and X and that t_0 is the identity. For example, t_α could be the group of affine transformations of $\mathcal{R}^2$:

$$t_\alpha\colon \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha_1 x + \alpha_2 y + \alpha_5 \\ \alpha_3 x + y + \alpha_4 y + \alpha_6 \end{pmatrix} \quad \text{with} \begin{vmatrix} 1 + \alpha_1 & \alpha_2 \\ \alpha_3 & 1 + \alpha_4 \end{vmatrix} \neq 0 \quad (41)$$

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

$$t_\alpha\colon \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x \cos\theta - y \sin\theta + a \\ x \sin\theta + y \cos\theta + b \end{pmatrix} \quad (42)$$

which is a Lie group with three parameters.

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

$$s(f, \alpha) = f \bigcirc t_\alpha^{-1} \quad (43)$$

This functional s, which takes another functional f as an argument, should remind the reader of Figure 2 where P, the discrete equivalent of f, is the argument of s.

The Lie algebra associated with the action of t_α on f is the space generated by the m local transformations L_{α_i} of f defined by:

$$L_{a_i}(f) = \left.\frac{\partial s(f, \alpha)}{\partial \alpha_i}\right|_{\alpha=0} \quad (44)$$

We can now write the local approximation of s as:

$$s(f, \alpha) = f + \alpha_1 L_{\alpha_1}(f) + \alpha_2 L_{\alpha_2}(f) + \ldots + \alpha_m L_{\alpha_m}(f) + o(\|\alpha\|^2)(f) \quad (45)$$

This equation is the continuous equivalent of Eq. (2) used in the introduction.

The following example illustrates how L_{α_i} can be computed from t_α. Consider the group of direct isometry defined in Eq. (42) (with parameter $\alpha = (\theta, a, b)$ as before, and $X = (x, y)$).

$$s(f, \alpha)(X) = f((x - a)\cos\theta + (y - b)\sin\theta,\ -(x - a)\sin\theta + (y - b)\cos\theta) \quad (46)$$

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

$$\frac{\partial s(f, \alpha)}{\partial \theta}(X) = y \frac{\partial f}{\partial x}(x, y) + (-x) \frac{\partial f}{\partial y}(x, y) \quad (47)$$

i.e.,

$$L_\theta = y \frac{\partial}{\partial x} + (-x) \frac{\partial}{\partial_y} \quad (48)$$

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

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:

$$s(f, \alpha) = f + \theta\left(y\frac{\partial f}{\partial x} + (-x)\frac{\partial f}{\partial y}\right) - a\frac{\partial f}{\partial x} - b\frac{\partial f}{\partial y} + o(\|\alpha\|^2)(f) \tag{49}$$

which corresponds to a linear combination of the three basic operators L_θ, L_a, and L_b.[5] The most important property is that the three operators generate the whole space of local transformations. The result of applying the operators to a function f, such as a 2-D image, is a set of vectors (referred to as tangent vector in the previous sections). Each point in the tangent space corresponds to a unique transformation. 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.

B. Tangent Vectors. The last problem to be solved is that 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 $\mathcal{R}$, which we assumed in Section IVA.

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

$$P'(x, y) = \sum_{i,j} P[i][j]\delta(x - i)\delta(y - j) \tag{50}$$

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:

$$C_\sigma: P \rightarrow f = P' * g_\sigma \tag{51}$$

In the 2-D case, the function f can be written as:

$$f(x, y) = \sum_{i,j} P[i][j]g_\sigma(x - i, y - j) \tag{52}$$

Other coding functions 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 sides of each pixels) is not a problem in practice; just choose one of the two values.

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

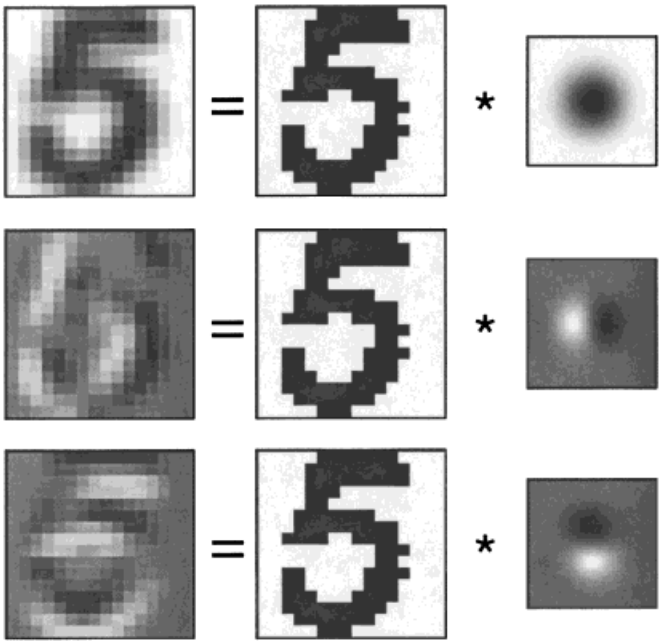

Figure 12. Graphic illustration of the computation of f and two tangent vectors corresponding to $L_x = \partial/\partial$x (X-tanslation) and $L_x = \partial/\partial$y (Y-translation), from a binary image I. The Gaussian function $g(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$ has a standard deviation of $\sigma = 0.9$ in this example although its graphic representation (small images on the right) have been rescaled for clarity.

The Gaussian mapping is preferred for two reasons. First, the smoothing parameter σ can be used to control the locality of the invariance. This is because when f is smoother, the local approximation of Eq. (45) is valid for larger transformations. 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:

$$L_X(f) = \frac{\partial}{\partial x}(P' * g_\sigma) = P' * \frac{\partial g_\sigma}{\partial x} \tag{53}$$

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 the Gaussian function g_σ (Fig. 12). Similarly, the tangent vector for scaling can be computed with:

$$L_S(f) = \left(x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}\right)(I * g_\sigma) = x\left(I * \frac{\partial g_\sigma}{\partial x}\right) + y\left(I * \frac{\partial g_\sigma}{\partial y}\right) \tag{54}$$

This operation is illustrated in Figure 13.

C. Important Transformations in Image Processing. This section summarizes how to compute the tangent vectors for image processing (in 2-D). Each discrete image I_i is convolved with a Gaussian of standard deviation g_σ to obtain a representation of the continuous image f_i, according to Eq. (55):

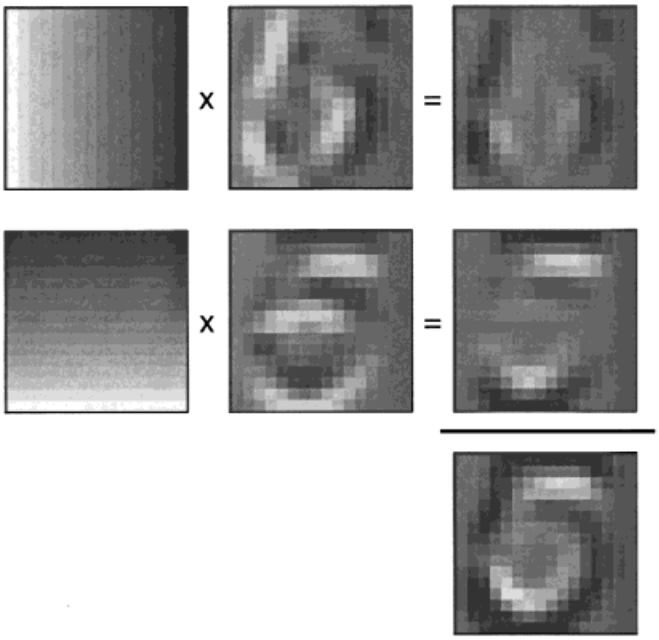

Figure 13. Graphic illustration of the computation of the tangent vector $T_u = D_x S_x + D_y S_y$ (bottom image). The displacement for each pixel is proportional to the distance of the pixel to the center of the image ($D_x(x, y) = x - x_0$ and $D_y(x, y) = y - y_0$). The two multiplications (horizontal lines) and the addition (vertical right column) are done pixel by pixel.

$$f_i = I_i * g_\sigma. \tag{55}$$

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 that is the tangent vector. The following list contains some of the most useful tangent vectors:

1. X-translation: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha \\ y \end{pmatrix} \tag{56}$$

The Lie operator is defined by:

$$L_X = \frac{\partial}{\partial x} \tag{57}$$

2. Y-translation: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x \\ y + \alpha \end{pmatrix} \tag{58}$$

The Lie operator is defined by:

$$L_Y = \frac{\partial}{\partial y} \tag{59}$$

3. Rotation: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x \cos \alpha - y \sin \alpha \\ x \sin \alpha + y \cos \alpha \end{pmatrix} \tag{60}$$

The Lie operator is defined by:

$$L_R = y \frac{\partial}{\partial x} + (-x) \frac{\partial}{\partial_y} \tag{61}$$

4. Scaling: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha x \\ y + \alpha y \end{pmatrix} \tag{62}$$

The Lie operator is defined by:

$$L_S = x \frac{\partial}{\partial x} + y \frac{\partial}{\partial y} \tag{63}$$

5. Parallel hyperbolic transformation: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha x \\ y - \alpha y \end{pmatrix} \tag{64}$$

The Lie operator is defined by:

$$L_S = x \frac{\partial}{\partial x} - y \frac{\partial}{\partial y} \tag{65}$$

6. Diagonal hyperbolic transformation: This transformation is useful when the classification function is invariant with respect to the input transformation:

$$t_\alpha: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha y \\ y + \alpha x \end{pmatrix} \tag{66}$$

The Lie operator is defined by:

$$L_S = y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y} \tag{67}$$

The resulting tangent vector is is the norm of the gradient of the image, which is easily computed.

7. Thickening: This transformation is useful when the classification function is invariant with respect to the variation of thickness. This is known in morphology as dilation and its inverse, erosion. It is useful in certain domains (such as handwritten character recognition) because thickening and thinning are natural variations that correspond to the pressure applied on a pen or to different absorbtion properties of the

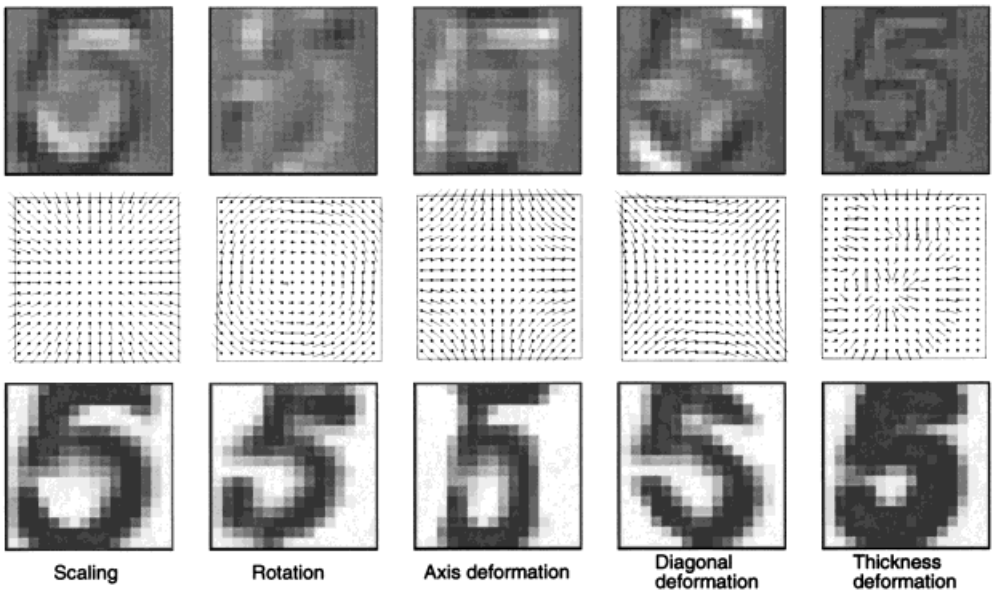

Figure 14. Illustration of five tangent vectors (top), corresponding displacements (middle), and transformation effects (bottom). The displacements D_x and D_y are represented in the form of a vector field. The tangent vector for the thickness deformation (right column) corresponds to the norm of the gradient of the gray-level image.

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 assume that the structural element is a sphere of radius α. We define the thickening transformation as the function that takes the function f and generates the function f'_α defined by:

$$f'_\alpha(X) = \max_{\|r\| \le \alpha} f(X + r) \quad \text{for } \alpha \ge 0 \tag{68}$$

$$f'_\alpha(X) = \max_{\|r\| \le -\alpha} f(X + r) \quad \text{for } \alpha \le 0 \tag{69}$$

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

$$\lim_{\alpha \to 0} \frac{f'(X) - f(X)}{\alpha} = \lim_{a \to 0} \frac{max_{\|r\| \le \alpha} f(X + r) - f(X)}{\alpha} \tag{70}$$

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

$$f(X + r) - f(X) = r \cdot \nabla f(X) + O(\|r\|^2) \approx r \cdot \nabla f(X) \tag{71}$$

The maximum of

$$\max_{\|r\| \le \alpha} f(X + r) - f(X) = \max_{\|r\| \le \alpha} r \cdot \nabla f(X) \tag{72}$$

is attained when r and $\nabla f(X)$ are colinear, that is, when

$$r = \alpha \frac{\nabla f(X)}{\|\nabla f(X)\|} \tag{73}$$

assuming $\alpha \ge 0$. It can easily be shown that this equation holds when α is negative, because we then try to minimize Eq. (69). Therefore:

$$\lim_{a \to 0} \frac{f'_\alpha(X) - f(X)}{\alpha} = \|\nabla f(X)\| \tag{74}$$

which is the tangent vector of interest. Note that this is true for α positive or negative. The same tangent vector describes both thickening and thinning. Alternatively, our computation of the displacement r can be used and the following transformation of the input can be defined as:

$$t_\alpha(f): \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x + \alpha r_x \\ y + \alpha r_y \end{pmatrix} \tag{75}$$

where

$$(r_x, r_y) = r = \alpha \frac{\nabla f(X)}{\|\nabla f(X)\|} \tag{76}$$

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:

$$L_T = \|\nabla\| \tag{77}$$

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 five transformations are depicted in Figure 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.

V. CONCLUSION

The basic tangent distance algorithm is easy to understand and implement. Even though hardly any preprocessing or learning is required, the performance is surprisingly good and compares well with 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 that performed better than tangent distance on one of the three databases was boosting, which has similar a priori knowledge about transformations by extending the training set with deformed data.

Many improvements are possible. For instance, smart preprocessing can allow measurement of the tangent distance in a more appropriate "feature" space, instead of in the original pixel space. For example, in image classification 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, to use a tangent distance. In this case, even the tangent vectors can be trained. The derivation has been done for batch training (Hastie and Simard, 1998) and for on-line training (Schwenk, 1998) of the tangent vectors. When such training is performed, the a priori knowledge comes from other constraints imposed on the tangent vectors (e.g., how many tangent vectors are allowed and which classes of transformation do they represent).

Finally, many optimizations commonly used in distance-based algorithms can also be used successfully with tangent distance to speed up computation. The multiresolution approach has already been tried successfully (Simard, 1994). Other methods like multiedit condensing (Devijver and Kittler, 1982; Voisin and Devijver, 1987) and K-d tree (Broder, 1990) 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, and predictions) 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 (LeCun et al., 1995). Fortunately, the concept of tangent vectors can also be used in learning. This is the basis for the tangent propagation algorithm. The concept is simple: instead of learning a classification function from examples of its values, one can also use information about its derivatives. This 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 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 sufficiently large, tangent distance or 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 (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.

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

ACKNOWLEDGMENTS

P. Simard and Y. LeCun gratefully acknowledge the National Science Foundation (NSF) for funding part of this work. A slightly edited version of this article also appears as a book chapter in *Neural Networks: Tricks of the Trade*, Springer-Verlag, Berlin pp 239–274, 1998.

REFERENCES

A.V. Aho, J.E. Hopcroft, and J.D. Ullman, Data structure and algorithms, Addison-Wesley, Reading, MA, 1983.

L. Bottou and V.N. Vapnik, Local learning algorithms, Neural Comput 4 (1992), 888–900.

A.J. Broder, Strategies for efficient incremental nearest neighbor search, Pattern Recog 23 (1990), 171–178.

D.S. Broomhead and D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Syst 2 (1988), 321–355.

Y. Choquet-Bruhat, C. DeWitt-Morette, and M. Dillard-Bleick, Analysis, manifolds and physics, North-Holland, Amsterdam, 1982.

C. Cortes and V. Vapnik, Support vector networks, Machine Learning 20 (1995), 273–297.

B.V. Dasarathy, Nearest neighbor (NN) norms: NN pattern classification techniques, IEEE Computer Society Press, Los Alamitos, CA, 1991.

P.A. Devijver and J. Kittler, Pattern recognition, a statistical approach, Prentice Hall, Englewood Cliffs, NJ, 1982.

H. Drucker, R. Schapire, and P.Y. Simard, Boosting performance in neural networks, Int J Pattern Recog Artif Intell 7 (1993), 705–719.

K. Fukunaga and T.E. Flick, An optimal global nearest neighbor metric, IEEE Trans Pattern Anal Machine Intell 6 (1984), 314–318.

R. Gilmore, Lie groups, lie algebras and some of their applications, Wiley, New York, 1974.

T. Hastie, E. Kishon, M. Clark, and J. Fan, A model for signature verification, Technical Report 11214-910715-07TM, AT&T Bell Laboratories, 600 Montain Ave, Murray Hill, NJ, 1992.

T. Hastie and P.Y. Simard, Metrics and models for handwritten character recognition, Stat Sci 13(1) (1998), 54–65.

T.J. Hastie and R.J. Tibshirani, Generalized linear models, Chapman and Hall, London, 1990.

G.E. Hinton, C.K.I. Williams, and M.D. Revow, "Adaptive elastic models for hand-printed character recognition," Advances in neural information processing systems, Morgan Kaufmann, Jan Mateo (Editor), 1992, pp. 512–519.

A.E. Hoerl and R.W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, Technometrics, 12 (1970), 55–67.

T. Kohonen, "Self-organization and associative memory," Springer-Verlag, Berlin, 3rd edition, 1989.

Y. LeCun, "Generalization and network design strategies," Connectionism in perspective, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels (Editors), Elsevier, Zurich, 1989, pp 143–155.

Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Handwritten digit recognition with a back-propagation network," Advances in neural information processing systems (vol. 2), D. Touretzky (Editor), Morgan Kaufman, Denver, 1990, pp 396–404.

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," Neural networks: The statistical mechanics perspective, J.H. Oh, C. Kwon, and S. Cho (Editors), World Scientific, 1995, pp. 261–276.

E. Parzen, On estimation of a probability density function and mode, Ann Math Stat 33 (1962), 1065–1076.

W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical recipes, Cambridge University Press, Cambridge, 1988.

H. Schwenk, The diabolo classifier, Neural Comput 10(8) (1998), 2175–2200.

R. Sibson, Studies in the robustness of multidimensional scaling: Procrustes statistics, J R Stat Soc 40 (1978), 234–238.

P.Y. Simard, "Efficient computation of complex distance metrics using hierarchical filtering," Advances in neural information processing systems, J. Cowan, G. Tesauro, and J. Alspector (Editors), Morgan Kaufman, San Francisco, CA, 1994, pp 168–175.

F. Sinden and G. Wilfong, On-line recognition of handwritten symbols, Technical Report 11228-910930-02IM, AT&T Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ, 1992.

V.N. Vapnik, Estimation of dependences based on empirical data, Springer-Verlag, New York, 1982.

V.N. Vapnik and A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, Theory Prob Applications 16 (1971), 264–280.

N. Vasconcelos and A. Lippman, "Multiresolution tangent distance for affine-invariant classification," Advances in neural information processing systems (vol. 10) M.I. Jordan, M.S. Kearns, U.S.A. Solla (Editors), MIT Press, Cambridge, 1998, pp. 843–849.

J. Voisin and P. Devijver, An application of the multiedit-condensing technique to the reference selection problem in a print recognition system, Pattern Recog 20 (1987), 465–474.