Chapter 17

The Manifold Perspective on Representation Learning

Manifold learning is an approach to machine learning that is capitalizing on the *manifold hypothesis* (Cayton, 2005; Narayanan and Mitter, 2010): *the data generating distribution is assumed to concentrate near regions of low dimensionality.* The notion of manifold in mathematics refers to continuous spaces that locally resemble Euclidean space, and the term we should be using is really *submanifold*, which corresponds to a subset which has a manifold structure. The use of the term manifold in machine learning is much looser than its use in mathematics, though:

- the data may not be strictly on the manifold, but only near it,
- the dimensionality may not be the same everywhere,
- the notion actually referred to in machine learning naturally extends to discrete spaces.

Indeed, although the very notions of a manifold or submanifold are defined for continuous spaces, the more general notion of *probability concentration* applies equally well to discrete data. It is a kind of informal *prior* assumption about the data generating distribution that seems particularly well-suited for AI tasks such as those involving images, video, speech, music, text, etc. In all of these cases the natural data has the property that *randomly choosing configurations of the observed variables according to a factored distribution (e.g. uniformly) are very unlikely to generate the kind of observations we want to model.* What is the probability of generating a natural looking image by choosing pixel intensities independently of each other? What is the probability of generating a meaningful natural language paragraph by independently choosing each character in a

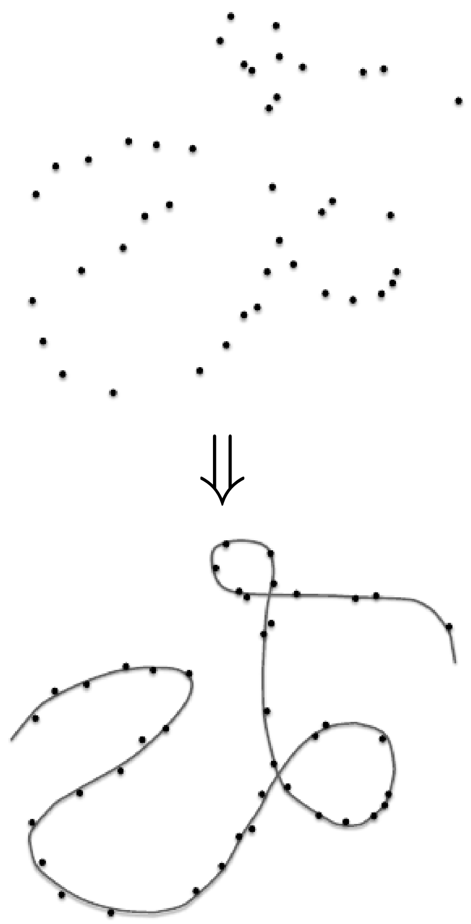

Figure 17.1: Top: data sampled from a distribution in a high-dimensional space (one 2 dimensions shown for illustration) that is actually concentrated near a one-dimensional manifold, which here is like a twisted string. Bottom: the underlying manifold that the learner should infer.

string? Doing a thought experiment should give a clear answer: an exponentially tiny probability. This is because the probability distribution of interest concentrates in a tiny volume of the total space of configurations. That means that to the first degree, the problem of characterizing the data generating distribution can be reduced to a binary classification problem: *is this configuration probable or not?*. Is this a grammatically and semantically plausible sentence in English? Is this a natural-looking image? Answering these questions tells us much more about the nature of natural language or text than the additional information one would have by being able to assign a precise probability to each possible sequence of characters or set of pixels. Hence, simply characterizing *where* probability concentrates is a fundamental importance, and this is what manifold learning algorithms attempt to do. Because it is a *where* question, it is more about *geometry* than about probability distributions, although we find both views useful when designing learning algorithms for AI tasks.

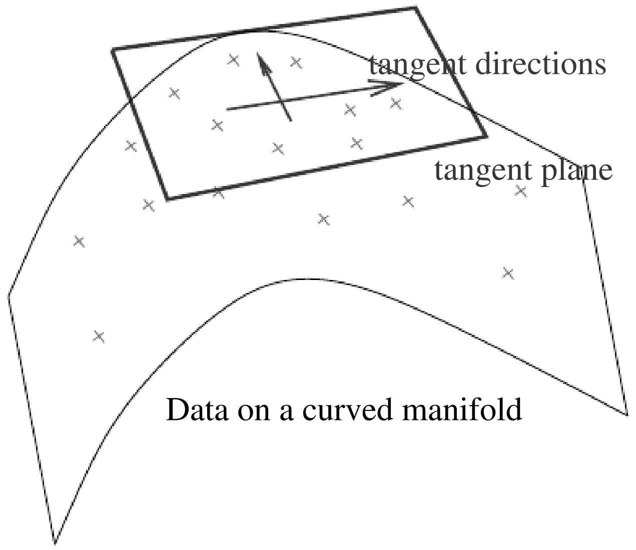

Figure 17.2: A two-dimensional manifold near which training examples are concentrated, along with a tangent plane and its associated tangent directions, forming a basis that specify the directions of small moves one can make to stay on the manifold.

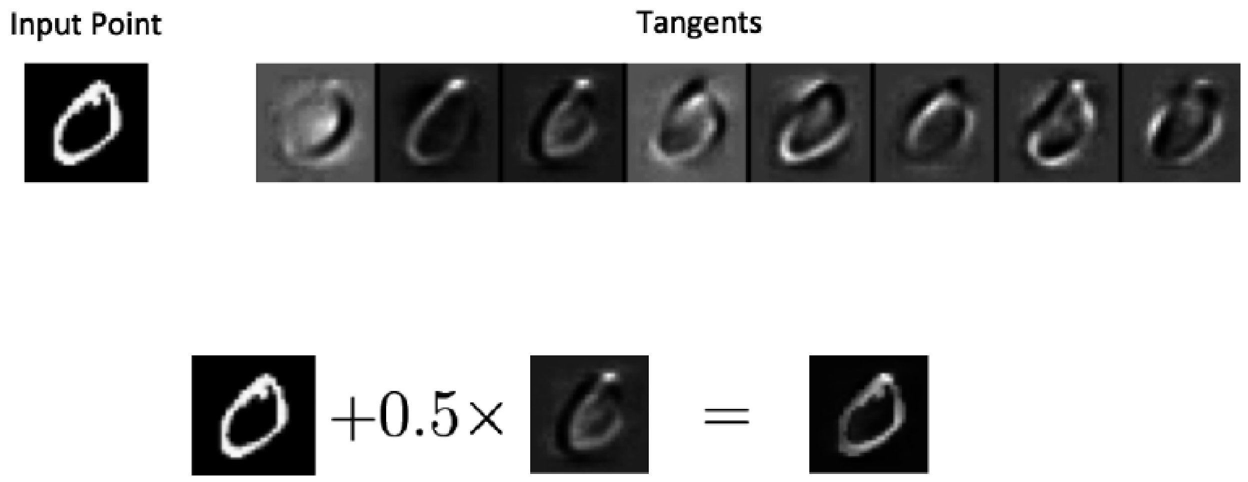

Figure 17.3: Illustration of tangent vectors of the manifold estimated by a contractive auto-encoder (CAE), at some input point (top left, image of a zero). Each image on the top right corresponds to a tangent vector. They are obtained by picking the dominant singular vectors (with largest singular value) of the Jacobian $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ (see Section 15.10). Taking the original image plus a small quantity of any of these tangent vectors yields another plausible image, as illustrated in the bottom. The leading tangent vectors seem to correspond to small deformations, such as translation, or shifting ink around locally in the original image. Reproduced with permission from the authors of Rifai *et al.* (2011a).

In addition to the property of probability concentration, there is another one that characterizes the manifold hypothesis: *when a configuration is probable it is generally surrounded (at least in some directions) by other probable configurations.* If a configuration of pixels looks like a natural image, then there are tiny changes one can make to the image (like translating everything by 0.1 pixel to the left) which yield another natural-looking image. The number of independent ways (each characterized by a number indicating how much or whether we do it) by which a probable configuration can be locally transformed into another probable configuration indicates the local *dimension of the manifold.* Whereas maximum likelihood procedures tend to concentrate probability mass on the training examples (which can each become a local maximum of probability when the model overfits), the manifold hypothesis suggests that good solutions instead concentrate probability along ridges of high probability (or their high-dimensional generalization) that connect nearby examples to each other. This is illustrated in Figure 17.1.

What is most commonly learned to characterize a manifold is a *representation* of the data points on (or near, i.e. projected on) the manifold. Such a representation for a particular example is also called its *embedding.* It is typically given by a low-dimensional vector, with less dimensions than the "ambient" space of which the manifold is a low-dimensional subset. Some algorithms (non-parametric manifold learning algorithms, discussed below) directly learn an embedding for each training example, while others learn a more general mapping, sometimes called an encoder, or representation function, that maps any point in the ambient space (the input space) to its embedding.

Another important characterization of a manifold is the set of its *tangent planes.* At a point x on a d-dimensional manifold, the tangent plane is given by d basis vectors that span the local directions of variation allowed on the manifold. As illustrated in Figure 17.2, these local directions specify how one can change x infinitesimally while staying on the manifold.

Manifold learning has mostly focused on unsupervised learning procedures that attempt to capture these manifolds. Most of the initial machine learning research on learning non-linear manifolds has focused on *non-parametric* methods based on the *nearest-neighbor graph.* This graph has one node per training example and edges connecting near neighbors. Basically, these methods (Schölkopf *et al.*, 1998; Roweis and Saul, 2000; Tenenbaum *et al.*, 2000; Brand, 2003; Belkin and Niyogi, 2003; Donoho and Grimes, 2003; Weinberger and Saul, 2004; Hinton and Roweis, 2003; van der Maaten and Hinton, 2008a) associate each of these nodes with a tangent plane that spans the directions of variations associated with the difference vectors between the example and its neighbors, as illustrated in Figure 17.4.

A global coordinate system can then be obtained through an optimization or

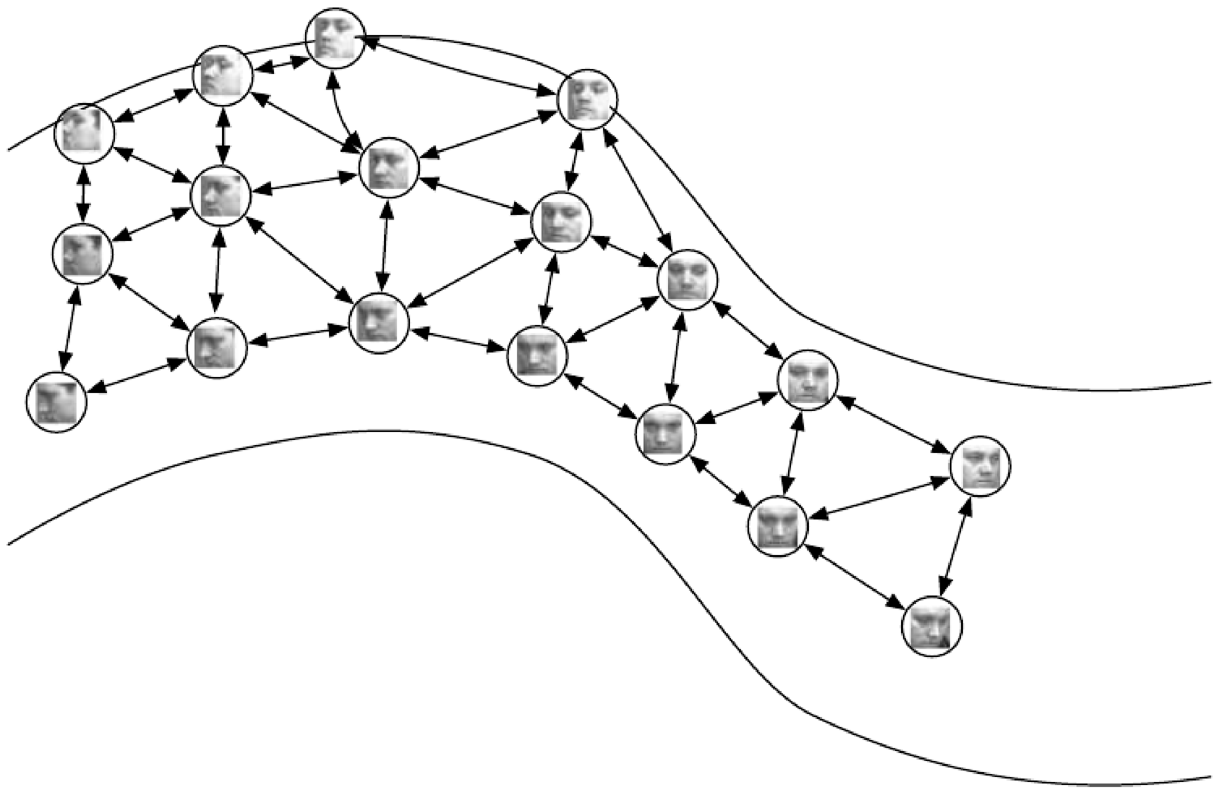

Figure 17.4: Non-parametric manifold learning procedures build a nearest neighbor graph whose nodes are training examples and arcs connect nearest neighbors. Various procedures can thus obtain the tangent plane associated with a neighborhood of the graph, and a coordinate system that associates each training example with a real-valued vector position, or *embedding*. It is possible to generalize such a representation to new examples by a form of interpolation. So long as the number of examples is large enough to cover the curvature and twists of the manifold, these approaches work well. Images from the QMUL Multiview Face Dataset (Gong *et al.*, 2000).

solving a linear system. Figure 17.5 illustrates how a manifold can be tiled by a large number of locally linear Gaussian-like patches (or "pancakes", because the Gaussians are flat in the tangent directions).

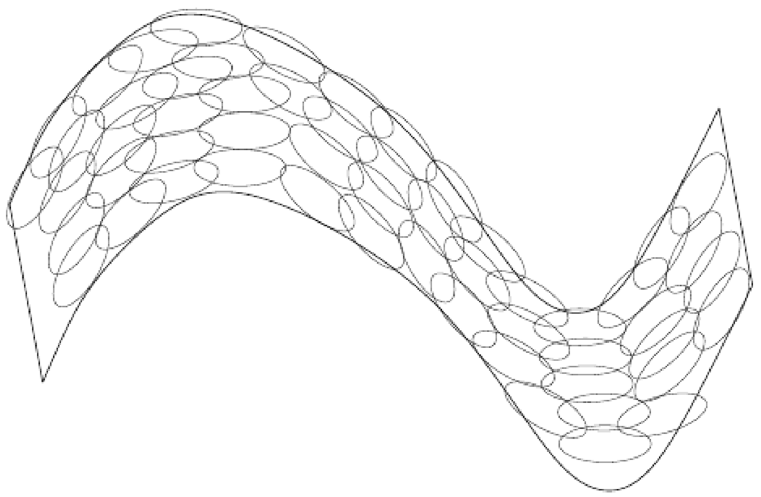

Figure 17.5: If the tangent plane at each location is known, then they can be tiled to form a global coordinate system or a density function. In the figure, each local patch can be thought of as a local Euclidean coordinate system or as a locally flat Gaussian, or *"pancake"*, with a very small variance in the directions orthogonal to the pancake and a very large variance in the directions defining the coordinate system on the pancake. The average of all these Gaussians would provide an estimated density function, as in the Manifold Parzen algorithm (Vincent and Bengio, 2003) or its non-local neural-net based variant (Bengio *et al.*, 2006c).

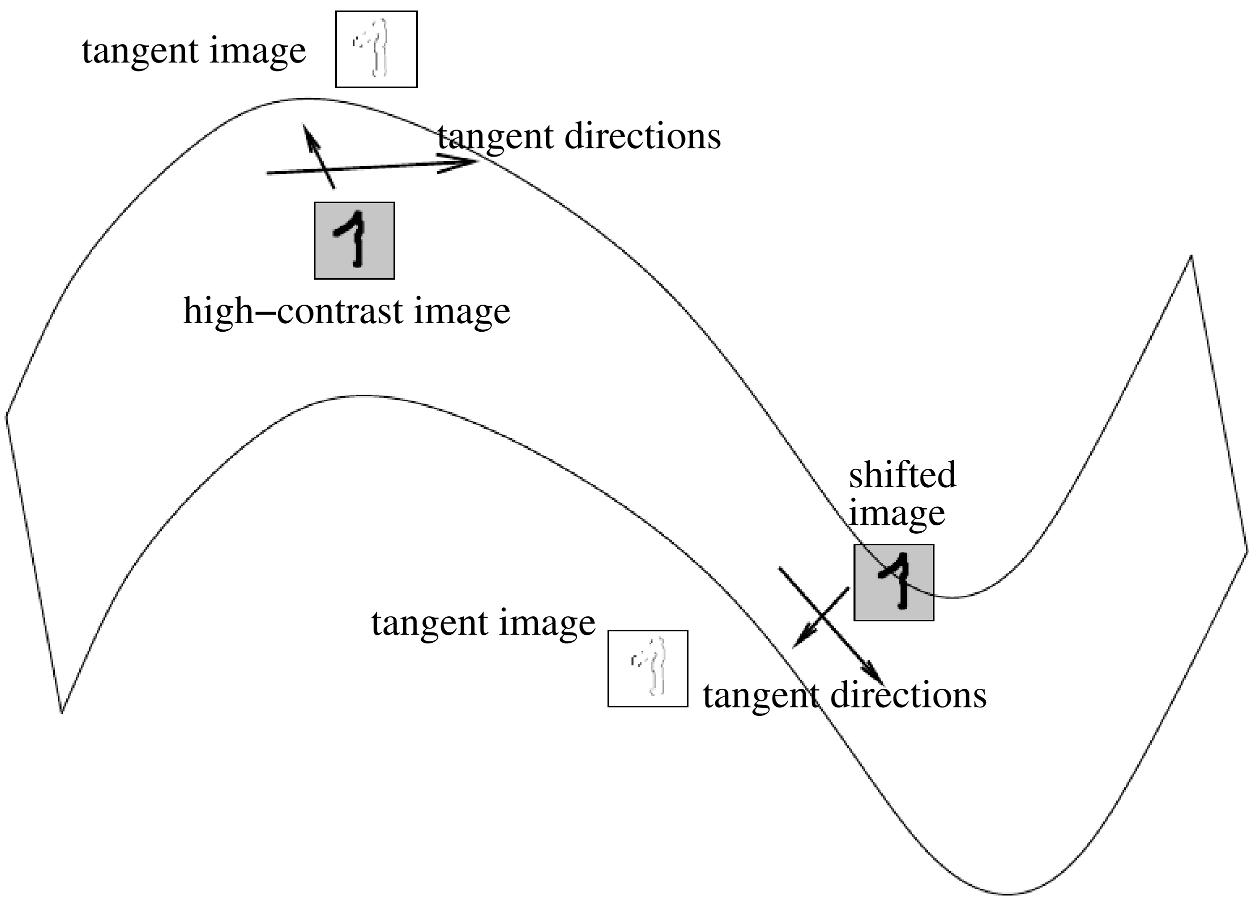

Figure 17.6: When the data are images, the tangent vectors can also be visualized like images. Here we show the tangent vector associated with translation: it corresponds to

However, there is a fundamental difficulty with such non-parametric neighborhood-based approaches to manifold learning, raised in Bengio and Monperrus (2005): if the manifolds are not very smooth (they have many ups and downs and twists), one may need a very large number of training examples to cover each one of these variations, with no chance to generalize to unseen variations. Indeed, these methods can only generalize the shape of the manifold by interpolating between neighboring examples. Unfortunately, the manifolds of interest in AI have many ups and downs and twists and strong curvature, as illustrated in Figure 17.6. This motivates the use of distributed representations and deep learning for capturing manifold structure, which is the subject of this chapter.

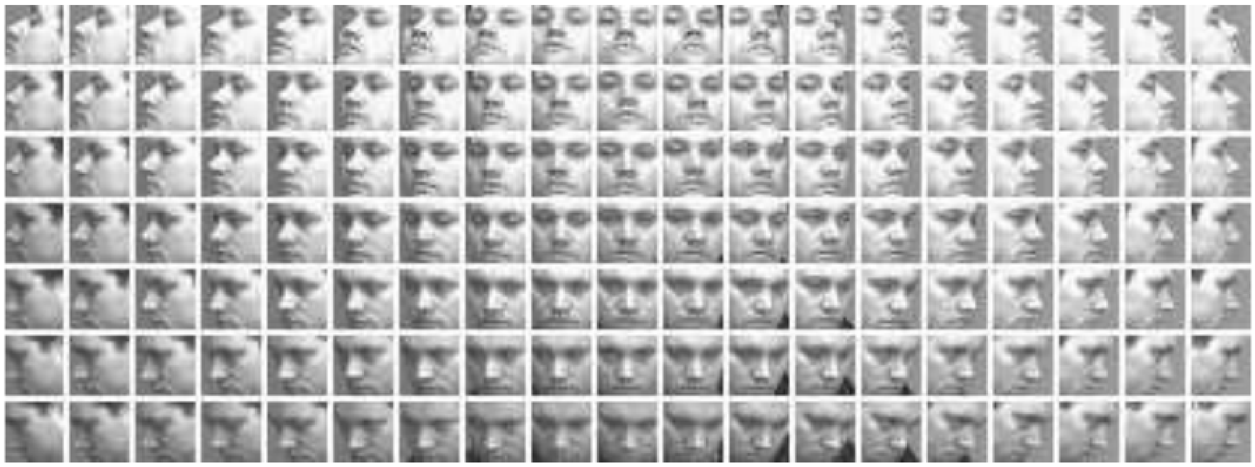
Figure 17.7: Training examples of a face dataset – the QMUL Multiview Face Dataset (Gong *et al.*, 2000) – for which the subjects were asked to move in such a way as to cover the two-dimensional manifold corresponding to two angles of rotation. We would like learning algorithms to be able to discover and disentangle such factors. Figure 17.8 illustrates such a feat.

The hope of many manifold learning algorithms, including those based on deep learning and auto-encoders, is that one learns an *explicit or implicit coordinate system* for the leading factors of variation that explain most of the structure in the unknown data generating distribution. An example of explicit coordinate system is one where the dimensions of the representation (e.g., the outputs of the encoder, i.e., of the hidden units that compute the "code" associated with the input) are directly the coordinates that map the unknown manifold. Training examples of a face dataset in which the images have been arranged visually on a 2-D manifold are shown in Figure 17.7, with the images laid down so that each of the two axes corresponds to one of the two angles of rotation of the face.

However, the objective is to *discover* such manifolds, and Figure 17.8 illustrates the images *generated* by a variational auto-encoder (Kingma and Welling, 2014a) when the two-dimensional auto-encoder code (representation) is varied on the 2-D plane. Note how the algorithm actually discovered two independent factors of variation: angle of rotation and emotional expression.

Another kind of interesting illustration of manifold learning involves the discovery of *distributed representations for words*. Neural language models were initiated with the work of Bengio *et al.* (2001c, 2003b), in which a neural network is trained to predict the next word in a sequence of natural language text, given the previous words, and where each word is represented by a real-valued vector, called *embedding* or *neural word embedding*.

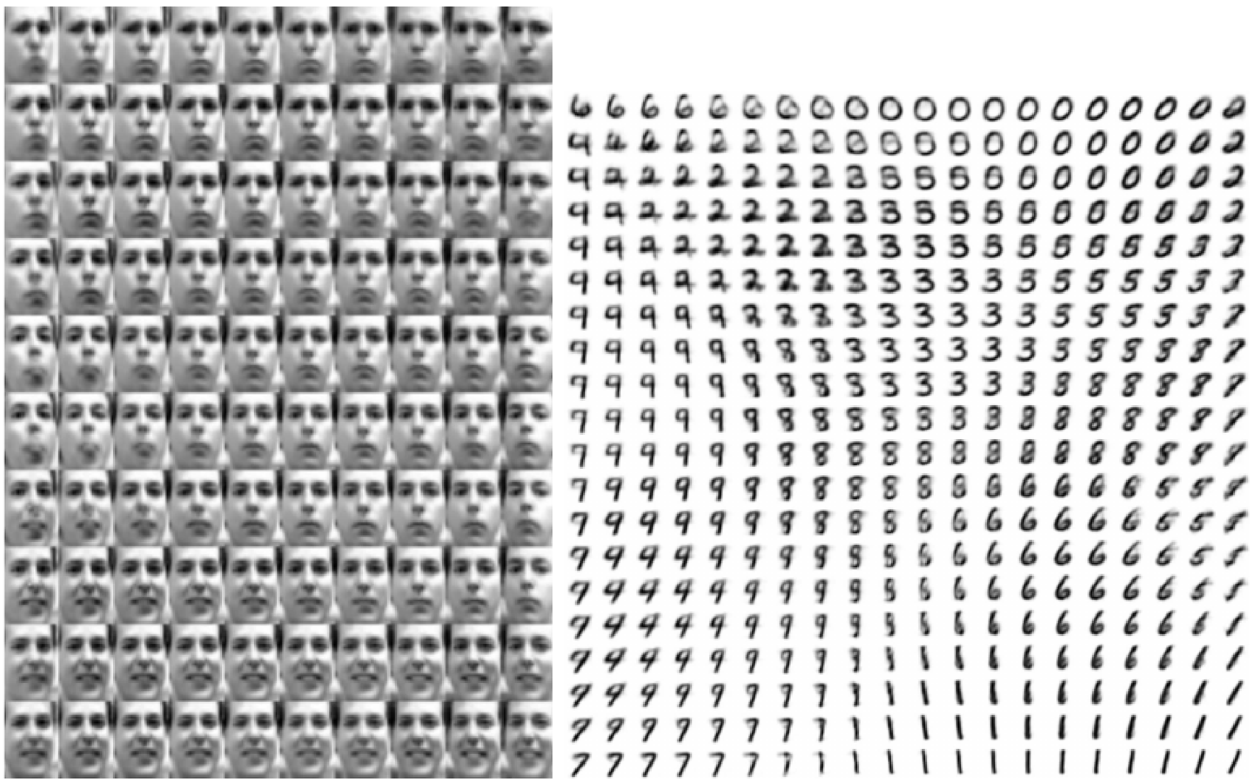

Figure 17.8: Two-dimensional representation space (for easier visualization), i.e., a Euclidean coordinate system for Frey faces (left) and MNIST digits (right), learned by a variational auto-encoder (Kingma and Welling, 2014a). Figures reproduced with permission from the authors. The images shown are not examples from the training set but images $\boldsymbol{x}$ actually generated by the model $P(\boldsymbol{x} \mid \boldsymbol{h})$, simply by changing the 2-D "code" $\boldsymbol{h}$ (each image corresponds to a different choice of "code" $\boldsymbol{h}$ on a 2-D uniform grid). On the left, one dimension that has been discovered (horizontal) mostly corresponds to a rotation of the face, while the other (vertical) corresponds to the emotional expression. The decoder deterministically maps codes (here two numbers) to images. The encoder maps images to codes (and adds noise, during training).

Figure 17.9 shows such neural word embeddings reduced to two dimensions (originally 50 or 100) using the t-SNE non-linear dimensionality reduction algorithm (van der Maaten and Hinton, 2008a). The figures zooms into different areas of the word-space and illustrates that words that are semantically and syntactically close end up having nearby embeddings.

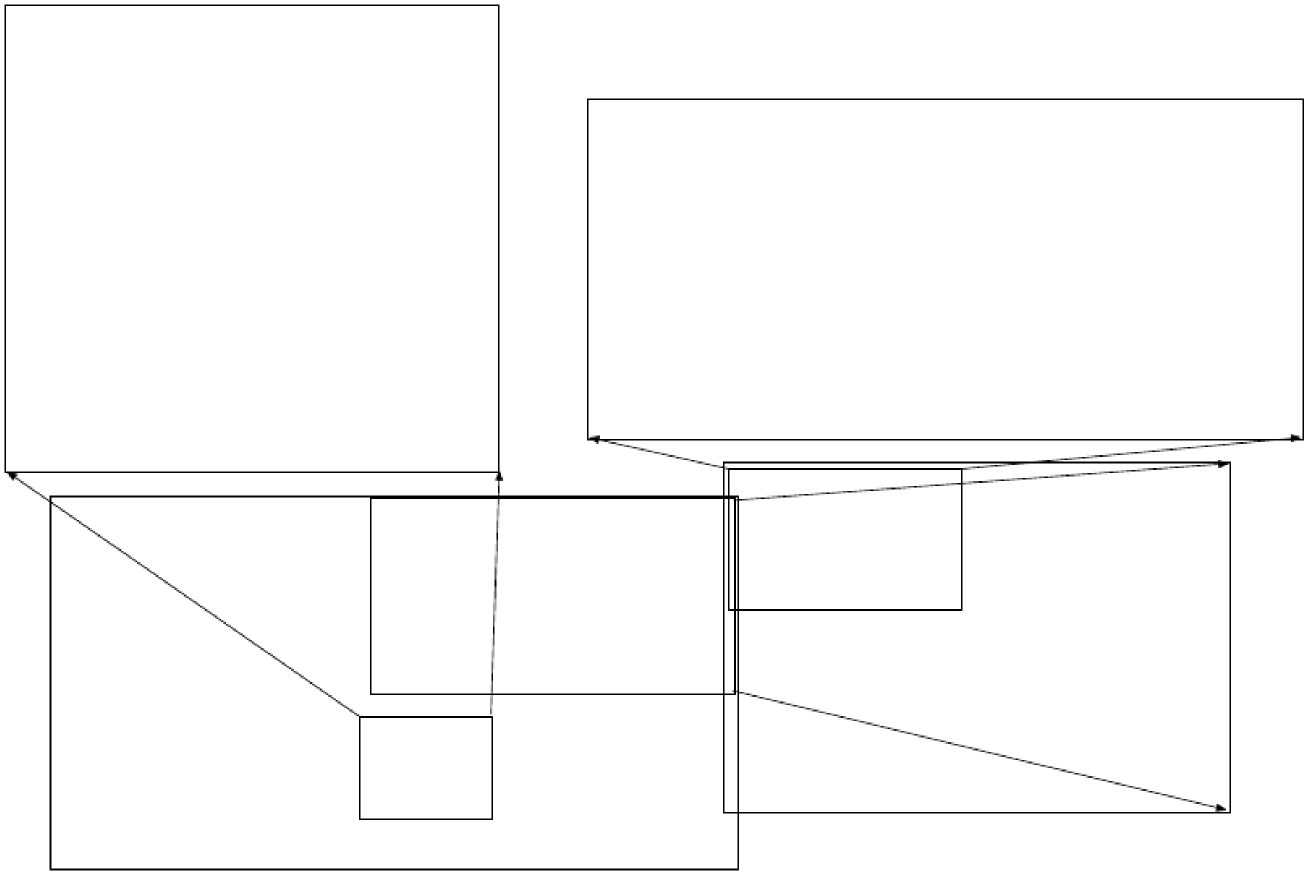

Figure 17.9: Two-dimensional representation space (for easier visualization), of English words, learned by a neural language model as in Bengio *et al.* (2001c, 2003b), with t-SNE for the non-linear dimensionality reduction from 100 to 2. Different regions are zoomed to better see the details. At the global level one can identify big clusters corresponding to part-of-speech, while locally one sees mostly semantic similarity explaining the neighborhood structure.

17.1 Manifold Interpretation of PCA and Linear Auto-Encoders

The above view of probabilistic PCA as a thin "pancake" of high probability is related to the manifold interpretation of PCA and linear auto-encoders, in which we are looking for projections of $\boldsymbol{x}$ into a subspace that preserves as much information as possible about $\boldsymbol{x}$. This is illustrated in Figure 17.10. Let the encoder be

$$\boldsymbol{h} = f(\boldsymbol{x}) = \boldsymbol{W}^{\top}(\boldsymbol{x} - \boldsymbol{\mu})$$

computing such a projection, a low-dimensional representation of h. With the auto-encoder view, we have a decoder computing the reconstruction

$$\hat{\boldsymbol{x}} = g(\boldsymbol{h}) = \boldsymbol{b} + \boldsymbol{V}\boldsymbol{h}.$$

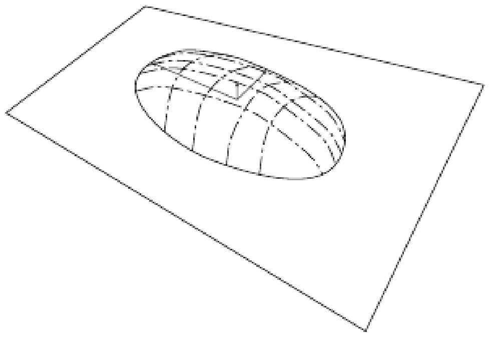

Figure 17.10: Flat Gaussian capturing probability concentration near a low-dimensional manifold. The figure shows the upper half of the "pancake" above the "manifold plane" which goes through its middle. The variance in the direction orthogonal to the manifold is very small (upward red arrow) and can be considered like "noise", where the other variances are large (larger red arrows) and correspond to "signal", and a coordinate system for the reduced-dimension data.

It turns out that the choices of linear encoder and decoder that minimize reconstruction error

$$\mathbb{E}[||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2]$$

correspond to $\boldsymbol{V} = \boldsymbol{W}$, $\boldsymbol{\mu} = \boldsymbol{b} = \mathbb{E}[\boldsymbol{x}]$ and the rows of $\boldsymbol{W}$ form an orthonormal basis which spans the same subspace as the principal eigenvectors of the covariance matrix

$$\boldsymbol{C} = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top].$$

In the case of PCA, the rows of $\boldsymbol{W}$ are these eigenvectors, ordered by the magnitude of the corresponding eigenvalues (which are all real and non-negative). This is illustrated in Figure 17.11.

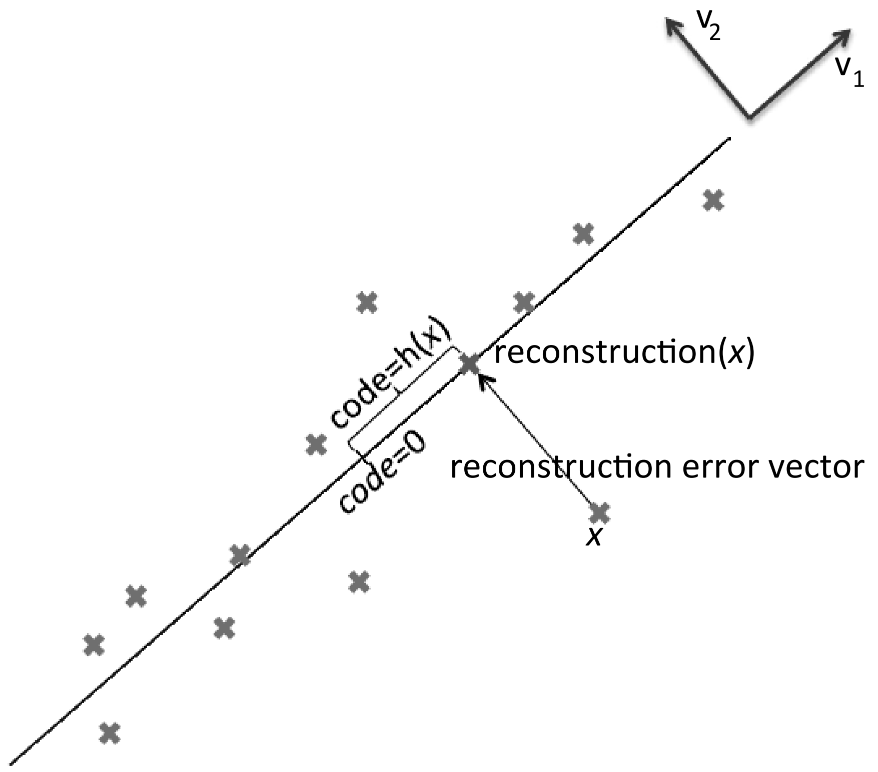

Figure 17.11: Manifold view of PCA and linear auto-encoders. The data distribution is concentrated near a manifold aligned with the leading eigenvectors (here, this is just $\boldsymbol{v}_1$) of the data covariance matrix. The other eigenvectors (here, just $\boldsymbol{v}_2$) are orthogonal to the manifold. A data point (in red, $\boldsymbol{x}$) is encoded into a lower-dimensional representation or code $\boldsymbol{h}$ (here the scalar which indicates the position on the manifold, starting from $\boldsymbol{h} = 0$). The decoder (transpose of the encoder) maps $\boldsymbol{h}$ to the data space, and corresponds to a point lying exactly on the manifold (green cross), the orthogonal projection of $\boldsymbol{x}$ on the manifold. The optimal encoder and decoder minimize the sum of reconstruction errors (difference vector between $\boldsymbol{x}$ and its reconstruction).

One can also show that eigenvalue λ_i of $\boldsymbol{C}$ corresponds to the variance of $\boldsymbol{x}$ in the direction of eigenvector $\boldsymbol{v}_i$. If $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{h} \in \mathbb{R}^d$ with $d < D$, then the optimal reconstruction error (choosing $\boldsymbol{\mu}$, $\boldsymbol{b}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ as above) is

$$\min \mathbb{E}[||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2] = \sum_{i=d+1}^{D} \lambda_i.$$

Hence, if the covariance has rank d, the eigenvalues λ_{d+1} to λ_D are 0 and reconstruction error is 0.

Furthermore, one can also show that the above solution can be obtained by maximizing the variances of the elements of $\boldsymbol{h}$, under orthonormal $\boldsymbol{W}$, instead of minimizing reconstruction error.

17.2 Manifold Interpretation of Sparse Coding

Sparse coding was introduced in Section 15.6.2 a linear factors generative model. It also has an interesting *manifold learning interpretation.* The codes $\boldsymbol{h}$ inferred with the above equation do not fill the space in which $\boldsymbol{h}$ lives. Instead, probability mass is concentrated on axis-aligned subspaces: sets of values of $\boldsymbol{h}$ for which most of the axes are set at 0. We can thus decompose $\boldsymbol{h}$ into two pieces of information:

- A binary pattern $\boldsymbol{\beta}$ which specifies which h_i are non-zero, with $N_a = \sum_i \beta_i$ the number of "active" (non-zero) dimensions.
- A variable-length real-valued vector $\boldsymbol{\alpha} \in \mathbb{R}^{N_a}$ which specifies the coordinates for each of the active dimensions.

The pattern $\boldsymbol{\beta}$ can be viewed as specifying an N_a-dimensional region in input space (the set of $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}$ where $h_i = 0$ if $b_i = 0$). That region is actually a linear manifold, an N_a-dimensional hyperplane. All those hyperplanes go through a "center" $\boldsymbol{x} = \boldsymbol{b}$. The vector $\boldsymbol{\alpha}$ then specifies a Euclidean coordinate on that hyperplane.

Because the prior $P(\boldsymbol{h})$ is concentrated around 0, the probability mass of $P(\boldsymbol{x})$ is concentrated on the regions of these hyperplanes near $\boldsymbol{x} = \boldsymbol{b}$. Depending on the amount of reconstruction error (output variance for $P(\boldsymbol{x} \mid g(\boldsymbol{h}))$), there is also probability mass bleeding around these hyperplanes and making them look more like pancakes. Each of these hyperplane-aligned manifolds and the associated distribution is just like the ones we associate to probabilistic PCA and factor analysis. The crucial difference is that instead of one hyperplane, we have 2^d hyperplanes if $\boldsymbol{h} \in \mathbb{R}^d$. Due to the sparsity prior, however, most of these flat Gaussians are unlikely: only the ones corresponding to a small N_a (with only a few of the axes being active) are likely. For example, if we were to restrict ourselves to only those values of $\boldsymbol{b}$ for which $N_a = k$, then one would have $\binom{d}{k}$ Gaussians. With this exponentially large number of Gaussians, the interesting thing to observe is that the sparse coding model only has a number of parameters linear in the number of dimensions of $\boldsymbol{h}$. This property is shared with other distributed representation learning algorithms described in this chapter, such as the regularized auto-encoders.

17.3 The Entropy Bias from Maximum Likelihood

TODO: how the log-likelihood criterion forces a learner that is not able to generalize perfectly to yield an estimator that is much smoother than the target distribution. Phrase it in terms of entropy, not smoothness.

17.4 Manifold Learning via Regularized Auto-Encoders

Auto-encoders have been described in Section 15. What is their connection to manifold learning? This is what we discuss here.

We denote f the encoder function, with $\boldsymbol{h} = f(\boldsymbol{x})$ the representation of $\boldsymbol{x}$, and g the decoding function, with $\hat{\boldsymbol{x}} = g(\boldsymbol{h})$ the reconstruction of $\boldsymbol{x}$, although in some cases the encoder is a conditional distribution $q(\boldsymbol{h} \mid \boldsymbol{x})$ and the decoder is a conditional distribution $P(\boldsymbol{x} \mid \boldsymbol{h})$.

What all auto-encoders have in common, when they are prevented from simply learning the identity function for all possible input $\boldsymbol{x}$, is that training them involves a compromise between two "forces":

1. Learning a representation $\boldsymbol{h}$ of training examples $\boldsymbol{x}$ such that $\boldsymbol{x}$ can be approximately recovered from $\boldsymbol{h}$ through a decoder. Note that this needs not be true for any $\boldsymbol{x}$, only for those that are probable under the data generating distribution.

2. Some constraint or regularization is imposed, either on the code $\boldsymbol{h}$ or on the composition of the encoder/decoder, so as to make the transformed data somehow simpler or to prevent the auto-encoder from achieving perfect reconstruction everywhere. We can think of these constraints or regularization as a preference for solutions in which the representation is as simple as possible, e.g., factorized or as constant as possible, in as many directions as possible. In the case of the bottleneck auto-encoder a fixed number of representation dimensions is allowed, that is smaller than the dimension of $\boldsymbol{x}$. In the case of sparse auto-encoders (Section 15.8) the representation elements h_i are pushed towards 0. In the case of denoising auto-encoders (Section 15.9), the encoder/decoder function is encouraged to be contractive (have small derivatives). In the case of the contractive auto-encoder (Section 15.10), the encoder function alone is encouraged to be contractive, while the decoder function is tied (by symmetric weights) to the encoder function. In the case of the variational auto-encoder (Section 20.9.3), a prior $\log P(\boldsymbol{h})$ is imposed on $\boldsymbol{h}$ to make its distribution factorize and concentrate as much as possible. Note how in the limit, for all of these cases, the regularization prefers representations that are insensitive to the input.

Clearly, the second type of force alone would not make any sense (as would any regularizer, in general). How can these two forces (reconstruction error on one hand, and "simplicity" of the representation on the other hand) be reconciled? The solution of the optimization problem is that *only the variations that are needed to distinguish training examples need to be represented.* If the data generating distribution concentrates near a low-dimensional manifold, this yields

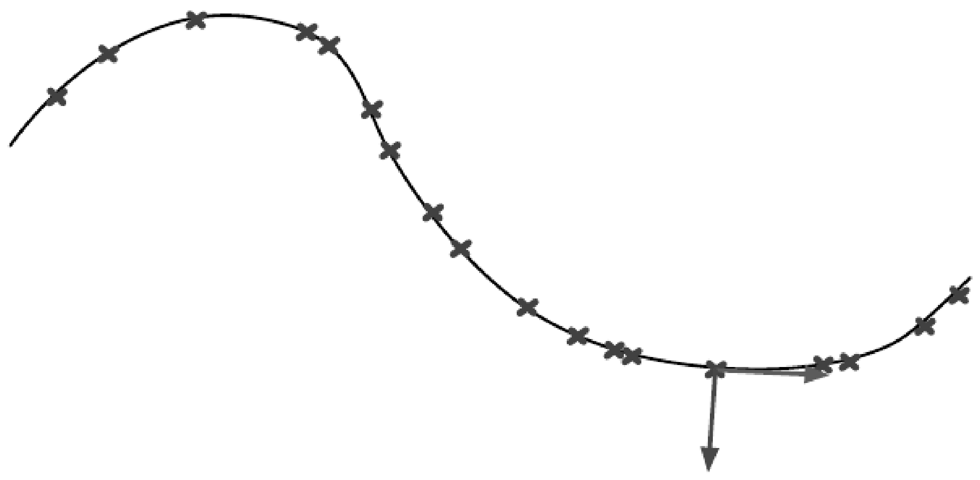

Figure 17.12: A regularized auto-encoder or a bottleneck auto-encoder has to reconcile two forces: reconstruction error (which forces it to keep enough information to distinguish training examples from each other), and a regularizer or constraint that aims at reducing its representational ability, to make it as insensitive as possible to the input in as many directions as possible. The solution is for the learned representation to be sensitive to changes along the manifold (green arrow going to the right, tangent to the manifold) but invariant to changes orthogonal to the manifold (blue arrow going down). This yields to *contraction* of the representation in the directions orthogonal to the manifold.

representations that implicitly capture a local coordinate for this manifold: only the variations tangent to the manifold around $\boldsymbol{x}$ need to correspond to changes in $\boldsymbol{h} = f(\boldsymbol{x})$. Hence the encoder learns a mapping from the embedding space $\boldsymbol{x}$ to a representation space, a mapping that is only sensitive to changes along the manifold directions, but that is insensitive to changes orthogonal to the manifold. This idea is illustrated in Figure 17.12. A one-dimensional example is illustrated in Figure 17.13, showing that by making the auto-encoder contractive around the data points (and the reconstruction point towards the nearest data point), we recover the manifold structure (of a set of 0-dimensional manifolds in a 1-dimensional embedding space, in the figure).

17.5 Tangent Distance, Tangent-Prop, and Manifold Tangent Classifier

One of the early attempts to take advantage of the manifold hypothesis is the Tangent Distance algorithm (Simard *et al.*, 1993, 1998). It is a non-parametric nearest-neighbor algorithm in which the metric used is not the generic Euclidean distance but one that is derived from knowledge of the manifolds near which probability concentrates. It is assumed that we are trying to classify examples

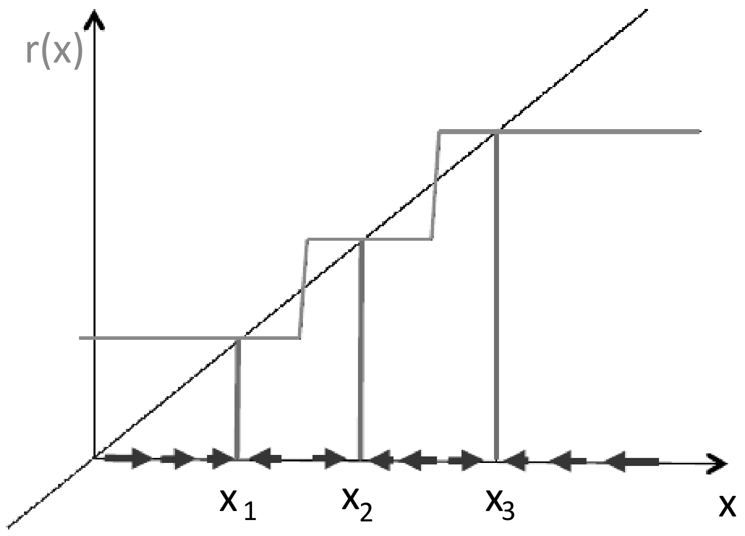

Figure 17.13: If the auto-encoder learns to be contractive around the data points, with the reconstruction pointing towards the nearest data points, it captures the manifold structure of the data. This is a 1-dimensional version of Figure 17.12. The denoising auto-encoder explicitly tries to make the derivative of the reconstruction function $r(\boldsymbol{x})$ small around the data points. The contractive auto-encoder does the same thing for the encoder. Although the derivative of $r(\boldsymbol{x})$ is asked to be small around the data points, it can be large between the data points (e.g. in the regions between manifolds), and it has to be large there so as to reconcile reconstruction error ($r(\boldsymbol{x}) \approx \boldsymbol{x}$ for data points $\boldsymbol{x}$) and contraction (small derivatives of $r(\boldsymbol{x})$ near data points).

and that examples on the same manifold share the same category. Since the classifier should be invariant to the local factors of variation that correspond to movement on the manifold, it would make sense to use as nearest-neighbor distance between points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ the distance between the manifolds M_1 and M_2 to which they respectively belong. Although that may be computationally difficult (it would require an optimization, to find the nearest pair of points on M_1 and M_2), a cheap alternative that makes sense locally is to approximate M_i by its tangent plane at $\boldsymbol{x}_i$ and measure the distance between the two tangents, or between a tangent plane and a point. That can be achieved by solving a low-dimensional linear system (in the dimension of the manifolds). Of course, this algorithm requires one to specify the tangent vectors at any point

In a related spirit, the Tangent-Prop algorithm (Simard *et al.*, 1992) proposes to train a neural net classifier with an extra penalty to make the output $f(x)$ of the neural net locally invariant to known factors of variation. These factors of variation correspond to movement of the manifold near which examples of the same class concentrate. Local invariance is achieved by requiring $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ to be orthogonal to the known manifold tangent vectors $\boldsymbol{v}_i$ at $\boldsymbol{x}$, or equivalently that the directional derivative of f at $\boldsymbol{x}$ in the directions $\boldsymbol{v}_i$ be small:

$$\text{regularizer} = \lambda \sum_i \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \cdot \boldsymbol{v}_i \right)^2 . \tag{17.1}$$

Like for tangent distance, the tangent vectors are derived a priori, e.g., from the formal knowledge of the effect of transformations such as translation, rotation, and scaling in images. Tangent-Prop has been used not just for supervised learning (Simard *et al.*, 1992) but also in the context of reinforcement learning (Thrun, 1995).

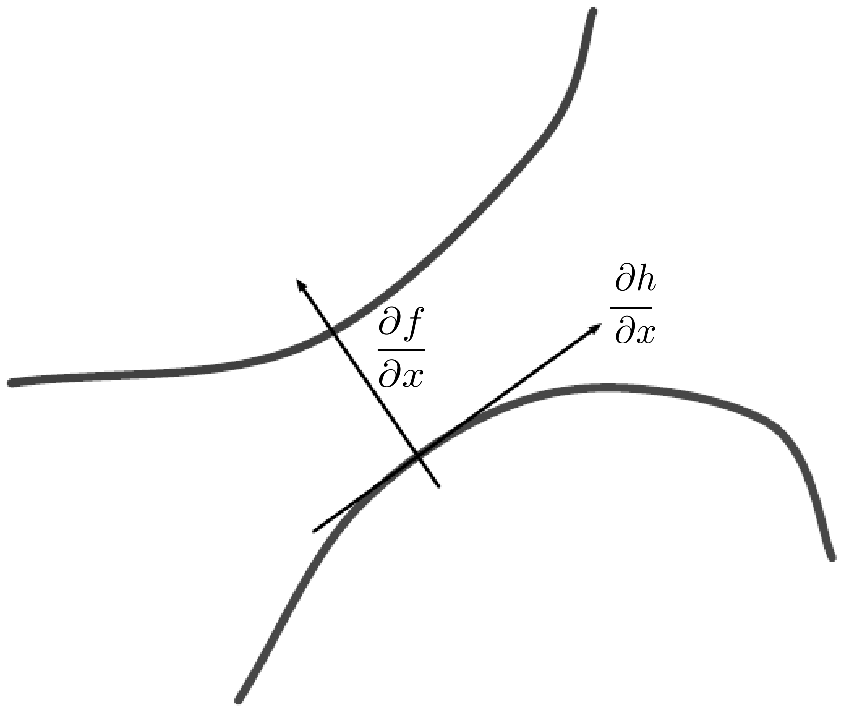
$\frac{\partial f}{\partial x}$
$\frac{\partial h}{\partial x}$

Figure 17.14: Illustration of the main idea of the tangent-prop algorithm (Simard *et al.*, 1992) and manifold tangent classifier (Rifai *et al.*, 2011d), which both regularize the classifier output function $f(\boldsymbol{x})$ (e.g. estimating conditional class probabilities given the input) so as to make it invariant to the local directions of variations $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ (manifold tangent directions). This can be achieved by penalizing the magnitude of the dot product of all the rows of $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ (the tangent directions) with all the rows of $\frac{\partial f}{\partial \boldsymbol{x}}$ (the directions of sensitivity of each output to the input). In the case of the tangent-prop algorithm, the tangent directions are given a priori, whereas in the case of the manifold tangent classifier, they are learned, with $\boldsymbol{h}(\boldsymbol{x})$ being the learned representation of the input $\boldsymbol{x}$. The figure illustrates two manifolds, one per class, and we see that the classifier output increases the most as we move from one manifold to the other, in input space.

A more recent paper introduces the Manifold Tangent Classifier (Rifai *et al.*, 2011d), which eliminates the need to know the tangent vectors a priori, and instead uses a contractive auto-encoder to estimate them at any point. As we have seen in the previous section and Figure 15.9, auto-encoders in general, and contractive auto-encoders especially well, learn a representation $\boldsymbol{h}$ that is most sensitive to the factors of variation present in the data $\boldsymbol{x}$, so that the leading singular vectors of $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ correspond to the estimated tangent vectors. As illustrated in Figure 15.10, these estimated tangent vectors go beyond the classical invariants that arise out of the geometry of images (such as translation, rotation and scaling) and include factors that must be learned because they are object-specific (such as adding or moving body parts). The algorithm proposed with the manifold tangent classifier is therefore simple: (1) use a regularized auto-encoder such as the contractive auto-encoder to learn the manifold structure by unsupervised learning (2) use these tangents to regularize a neural net classifier as in Tangent-Prop (Eq. 17.1). TODO Tangent Prop or Tangent-Prop?