Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

Learning high-dimensional data

Michel VERLEYSEN
Université catholique de Louvain, Microelectronics laboratory
3 place du Levant, B-1348 Louvain-la-Neuve, Belgium
e-mail: verleysen@dice.ucl.ac.be

Abstract. Observations from real-world problems are often high-dimensional vectors, i.e. made up of many variables. Learning methods, including artificial neural networks, often have difficulties to handle a relatively small number of high-dimensional data. In this paper, we show how concepts gained from our intuition on 2- and 3-dimensional data can be misleading when used in high-dimensional settings. When then show how the "curse of dimensionality" and the "empty space phenomenon" can be taken into account in the design of neural network algorithms, and how non-linear dimension reduction techniques can be used to circumvent the problem. We conclude by an illustrative example of this last method on the forecasting of financial time series.

1. Introduction

In the last few years, data analysis, or data mining, has become a specific discipline, sometimes far from its mathematical and statistical origin. Analyzing data is a real scientific job, where experience and thinking is often more valuable than using mathematical theorems and statistical criteria.

The specificity of modern data mining is that *huge* amounts of data are considered. Compared to just a few years ago, we now use daily huge databases in medical research, imaging, financial analysis, and many other domains. Not only new fields are open to data analysis, but also it becomes easier, and cheaper, to collect large amounts of data.

One of the problems related to this tremendous evolution is the fact that analyzing these data becomes more and more difficult, and requires new, more adapted techniques than those used in the past. A main concern in that direction is the *dimensionality* of data. Think of each measurement of data as one observation, each observation being composed of a set of variables. It is very different to analyze 10000 observations of 3 variables each, than analyzing 100 observations of 50 variables each! One way to get some feeling of this difficulty is to imagine each observation as a point in a space whose dimension is the number of variables. 10000 observations in a 3-dimensional space most probably form a structured shape, one or several clouds, from which it is possible to extract some relevant information, like principal directions, variances of clouds, etc. On the contrary, at first sight 100 observations in a 50-dimensional space do not represent anything specific,

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

because the number of observations is too low. Imagine that one would like to extract information from 4 observations in a 3-dimensional space...

Nevertheless, many modern databases *have* this unpleasant characteristic. Should we conclude that nothing has to be done, and that data mining is just impossible in that situation? Of course not. First because it is the duty of scientists to invent methods adapted to real, existing problems (and not to invent unrealistic problems that "prove" the validity of their results...). Secondly because even in such situation, there *are* ways to analyze the data, to extract information and to draw conclusions from observations. The point is that the methods are different, and sometimes more related to ad-hoc but sounded algorithms than to mathematical theorems not valid in this context.

This paper makes no pretence of presenting generic solutions to this problem; the current state-of-the-art is far from that. However, we will try to convince the reader about the importance of this topic, not only in data mining, but also in function approximation, identification, forecasting, i.e. all domains where the learning abilities of artificial neural networks are exploited. We will show through a few examples how distributions of points in high-dimensional spaces can behave in a drastically different way from our intuition (mostly gained from 2- and 3-dimensional schemes). Finally, we will show how projection tools could be used, not to remove the difficulties related to high-dimensional data, but to circumvent them by working with lower-dimensional representations.

2. High-dimensional data are difficult to use

Many references dealing with the problems and difficulties related to the use of high-dimensional data exist in the scientific literature. Unfortunately, most of them are somewhat hopeless descriptions of the problems, rather than invitations to a new, fascinating challenge for 21st-century data analysts. I recently discovered the "aide-mémoire" written by David Dohonen for his lecture to the "Mathematical Challenges of the 21st Century" conference of the American Mathematical Society. Despite his lecture is not oriented (or biased...) towards artificial neural networks, the accompanying text is one of the most fascinating ones I had the opportunity to read on this topic recently [1]. The following of this section contains several ideas borrowed from Dohonen's paper.

2.1. Data

Modern data analysis deals with considerable amounts of information. Here are a few examples of domains where collecting information on a large scale opens new lines of research or exploitation.

- Medical data are now collected widely. As an example, digital imaging now makes possible the storage, and the analysis, of vast databases of radiology images. This not only helps the patient, whose medical files are better archived and forwarded between hospitals and doctors; it also contributes to a better understanding of pathologies, through the building-up and comparison of large databases of healthy and unhealthy patients. EEG recordings are another example in this context.
- Investors and traders, in the hope of discovering some helpful laws or trends, analyze financial series continuously; databases with high-frequency values of financial indices are now publicly available.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

- All purchases made by credit or payment card are now stored in order to analyze the customer's habits; this helps to the detection of credit card fraud, but also to identify the consumers habits and to target commercial campaigns...

Of course, this list is far from being exhaustive. It serves only to make clear that we are now dealing with *huge* databases, and that elements (or *observations*) in these databases may take different form: scalar values (a single financial index), vectors (simultaneous recording of 40 EEG signals), images, etc.

2.2. Size of data

What we mean by "huge databases" is the fact that many observations are collected. But what about the size of these observations? In other words, how many variables are typically gathered in a single observation? The above examples show a wide variety of answers from one or a few to hundreds or thousands: when an observation is an image, each image is a point in a space whose dimension is the number of pixels! In general, a database has n lines and d columns, where n is the number of observations (which can increase over time) and d is the number of observed variables.

What makes modern data analysis different from traditional mathematical statistics is the fact that we are dealing with very different values for n and d. In addition to larger values, it is the relation between n and d that differs the most sensibly. Think for example to PCA (Principal Component Analysis). Most textbooks explain PCA as a technique to find directions in a cloud of points, and illustrate the method on a 2-dimensional figure with several hundreds of points (thus n = several hundreds, and $d = 2$). But PCA is evenly used in face identification tasks [2] where an observation is an image (d = several hundreds) and the database only contains a few observations! Traditional mathematical statistics is clearly oriented towards situations where $n >> d$. Worst, some results and tools are derived from asymptotic situations where $n \to \infty$ with d fixed, while in reality we could have d increasing with n fixed!

One of the reasons of increasing d can also be the lack of information about the usefulness of variables. Think for example to the prediction of financial data. It is easy to imagine that the fluctuations of a specific share are largely influenced by the recent fluctuations of stock markets. Using current values of several stock markets as exogenous input variables to a forecasting method, whose output is the share value to predict, is then a standard way of working. Unfortunately, we do not know which stock market (or other...) indices to use, so we are tempted to use as many of them as possible! This increases d (for n fixed), which is non-traditional in terms of statistical analysis.

Another reason why modern data analysis differs from traditional statistics is that in the latter, many nice results make the assumption that data are samples from multivariate normal distributions. As a consequence, tools from linear algebra may be used, and strongly simplify some computations. However, except in some specific situations, multivariate normal distributions are far from being natural; when the dimension d of the space is large, this is further reinforced by the fact that even the central limit theorem cannot be used because of a too small number n of samples (with respect to d).

Data analysis, and other tasks performed by neural networks, must be considered on the “engineering” point of view: we must work with the data we have, with their dimensionality, and adapt the methods we use to their characteristics. Even if it seems hard at first glance!

Up to now, we intentionally did not discuss a lot about artificial neural networks. Indeed when speaking about high dimensions, most of the problems do not come from the use of

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

neural networks, nor from any other competing method. The problems are related to the nature itself of the data: using neural networks or other methods does not change anything. The type of task to achieve is not crucial either: whatever we speak about function approximation, clustering, classification, or (self-)organization of data, we always face the same difficulties if we have a rather small number of high-dimensional data available for learning. Nevertheless, we will try to show in the following that we can take advantage from some neural network techniques to fight successfully against the difficulties related to high-dimensional data.

2.3. The curse of dimensionality

According to [1], Richard Bellman probably invented the phrase "the curse of dimensionality" [3]. Bellman discussed the simple problem of optimizing a function of a few dozens of variables, by exhaustive search in the function domain. If we consider a function defined on the unit cube in dimension d, and if ten discrete points are considered in each dimension, we need 10^d evaluations of the function, which is computationally prohibitive even for moderate d; Bellman used this argument to suggest dynamic programming instead of exhaustive search for function optimization.

More specifically, according to [1], the curse of dimensionality may be seen as follows: in function approximation and optimization, without assumption on the function of d variables to approximate or optimize, we need order $(1/\varepsilon)^d$ evaluations of the function on a grid to obtain an approximation (respectively optimization) with error ε.

We will see in the following that a way to circumvent the curse of dimensionality is to make assumptions on the function to approximate (or to optimize). More specifically, let us have a look again to the problem of forecasting the value of a share, using its past values and stock market indexes as exogenous input variables. As we do not know exactly which exogenous variables we have to use, a conservative way is to use many of them, leading to a high-dimensional problem. Nevertheless, all variables used are not independent one from another: there *is* an influence from each stock market index on all other ones. The information contained in the set of exogenous variables if thus redundant, the problem being that we do not know the characteristics of this redundancy. But the consequence of this is that some of the variables of the input space are dependent from other variables; even if we are in a d-dimensional input space, the points could thus be situated on a m-dimensional surface or submanifold, with $m < d$. The real dimensionality of the problem is m, even if a d-dimensional space is used. In the following, we will see that this concept is related to the *intrinsic* dimensionality of the data, and we will extensively use this property in order to circumvent the curse of dimensionality.

For the sake of completeness, we have to mention that other types of assumptions are also possible. For example, a famous result from Barron [4] shows that it is possible to approximate a function with a rate that is independent from the dimension, provided some specific assumptions about the function itself are met. While this and other results are of importance when dealing with the curse of dimensionality, we concentrate in this paper on assumptions made on data rather than on the function to approximate.

Finally, it is also important to mention that high-dimensional spaces also have their advantages. Donoho [1] mentions a few "blessings of dimensionality" that can be exploited when working with high-dimensional data. The blessings include the so-called "concentration of measure" (the variance of any measure –distance, norm– remains fixed while its norm increases with the dimension of the space), and asymptotic results that can be derived when the dimension increases to infinity.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

3. Surprising results in high-dimensional spaces

Intuitive comments in the previous section showed that more learning points are needed in higher-dimensional settings. But two questions arise at this stage:

- what is the limit between low- and high-dimensional spaces?
- what is the number of points required for learning in high-dimensional spaces?

The following subsections give some elements of information about these two questions.

3.1. Limit between low- and high-dimensional data

Textbooks and scientific articles illustrate learning methods and other data analysis tools on one-, two- or three-dimensional examples, and measure their performances on higher-dimensional problems for which no representation is possible. It is also widely accepted that most real-world problems are high-dimensional ones. But where is the limit between small and high dimension?

To answer to this question, let us take a few examples of concepts that are both intuitive in dimensions up to 3 and expandable to larger ones. The dimension where our intuitive view is not valid anymore is the answer to our question.

Scott and Thompson [5] first noticed the problems related to data in high dimensions, and called them "empty space phenomenon".

Consider the volume of a sphere in dimension d. This volume is given by

$$V(d) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d \tag{1}$$

where r is the radius, or equivalently by the recurrence equation

$$V(d) = V(d-2)\frac{\pi}{d-2} r^2 \tag{2}$$

with $V(1) = 2$ and $V(2) = \pi$. Looking at the graph of $V(d)$ when $r = 1$ (Figure 1) leads to the surprising observation that the volume rapidly decreases towards 0 when d increases!

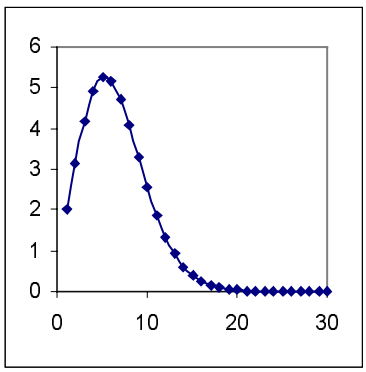

Figure 1: volume of the sphere (radius = 1) versus the dimension of the space.

Figure 1 is the standard graph found in the literature to illustrate the volume of the sphere. Nevertheless, it must be reminded that our intention here is to show that our

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

intuitive view of the volume of a sphere is misleading in high dimension. We should thus compare this volume to a value that seems "natural" to us. One way to do this is to plot the ratio between the volume of a sphere and the volume of a cube (with edge length equal to the diameter of the sphere). Figure 2 shows this ratio.

Having in mind a segment, a circle and a sphere respectively in dimension one, two and three, we understand that the ratio illustrated in Figure 2 will decrease with the dimension of the sphere. What is more surprising is that this ratio is below 10% when the dimension is as low as 6!

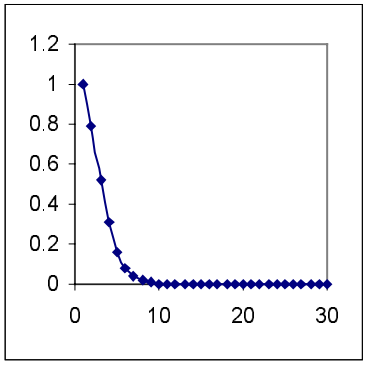

Figure 2: ratio between the volume of a sphere and the volume of a cube (length of an edge equal to the diameter of the sphere) versus the dimension of the space.

Another way to consider this problem is to plot (Figure 3) the ratio between the volume of a sphere with radius 0.9 and a sphere with radius 1, versus the dimension. Obviously, this ratio is equal to 0.9 raised to the power *d*. The values plotted in Figure 3 mean that 90% of the volume of a sphere in dimension greater than 20 is contained in the spherical shell whose thickness is 10% of the initial radius!

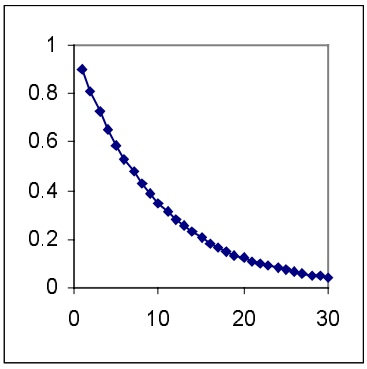

Figure 3: ratio between the volume of a sphere with radius 0.9 and the volume of a sphere with radius 1, versus the dimension of the space.

Another comment concerns Gaussian functions in high dimensions. We will see in the following of this paper that we advocate the use of Gaussian kernels as function approximators (Radial-Basis Function Networks with Gaussian kernels). But what is a Gaussian function in high dimension? Intuitively Gaussian functions are used for their local properties: most of the integral the function is contained in a limited volume around its centre. It is well known that 90% of the samples of a normalized scalar Gaussian distribution fall statistically in the interval [-1.65, 1.65]. What is less obvious is that this percentage rapidly decreases to 0 with the dimension of the space! Figure 4 shows the

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

percentage of samples of a Gaussian distribution falling in the sphere of radius 1.65, versus the dimension of the space. In dimension 10 already this percentage is below 1%!

In other words, when the dimension increases, most of the volume of a Gaussian function is contained in the tails instead of near the centre! This suggests that a Gaussian function could not be appropriate in high dimensions, at least regarding the local character mentioned above. A solution could be to use super-Gaussian functions, as for example the kernels considered in [6].

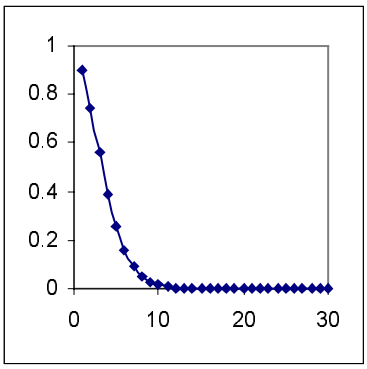

Figure 4: percentage of samples from a Gaussian distribution falling in the sphere of radius 1.65, versus the dimension of the space.

3.2. Number of learning points in high-dimensional settings

Our discussion about the problems related to high dimensions is also intended to point out to the necessity for many samples (and even more...) when the dimension increases. It is difficult to tackle the problem of finding the number of samples required to reach a predefined level of precision in approximation tasks. The reason is that results tackling levels of precision in approximation tasks are usually rather theoretical ones, derived from asymptotic developments where the number of samples tends to infinity.

Intuitively, we understand that this number could increase exponentially with the dimension of the space. Reminding Bellman's argument cited in the previous section, we note that if 10 learning points are necessary to obtain a defined precision in a scalar function approximation problem (i.e. if 10 discrete learning points are considered), the same level of precision would require 10^d learning points in a d-dimensional setting.

As an example, Silverman [7] addressed the problem of finding the required number of samples for a specific problem in the context of the approximation of a Gaussian distribution with fixed Gaussian kernels. Silverman's results are summarized in Figure 5.

Silverman's results can be approximated by [6]

$$\log_{10} N(d) \cong 0.6(d - 0.25). \tag{3}$$

In practice, any data set that does not grow exponentially with the dimension of the space will be referred to as *small*, or conversely, the dimension will be said to be *large*.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

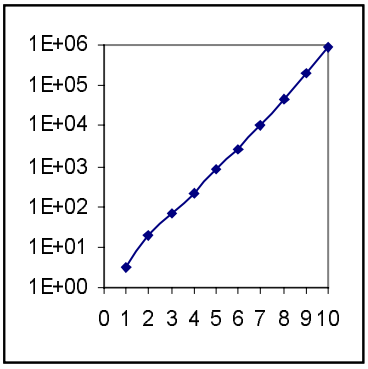

Figure 5: Number of samples required to approximate a Gaussian distribution with fixed Gaussian kernels, with an approximate error of about 10% (according to [6]), versus the dimension of the space.

This section will be concluded by an important comment. The above discussion could make the reader think that one never has enough samples in high dimension. Consider indeed a problem where 10 samples would be required in dimension 1 and 100 in dimension 2 (imagine a segment and a square "filled" by respectively 10 and 100 samples). This would mean that 10^{10} and 10^{20} samples would be required in dimensions 10 and 20 respectively! This is obviously impossible in real-world problems (while dimensions much greater than 20 are common). Having (apparently) too few points for an efficient learning is called the “empty space phenomenon”.

Fortunately, as already mentioned in the previous section, it appears that real-world problem do not suffer so severely from the "curse of dimensionality" problem (while its importance should not be neglected tough). This is simply due to the fact that in most situations, data are located near a manifold of dimension *m* smaller than *d*. Reducing the dimension of the data to a smaller value than *d*, in order to decrease the problems related to high dimensions, is a key issue in high-dimensional data learning.

3.3. Concentration of measure phenomenon

While this is not directly related to the two questions introducing this section, the “concentration of measure phenomenon” must be mentioned here, as another surprising result in high-dimensional spaces. Donoho [1] mentions this phenomenon in his list “blessings of dimensionality”, meaning that advantage could be gained from it. We prefer to report this phenomenon as something that the data analyst should be aware of, in order to benefit from its advantages but also to take care of its drawbacks.

The “concentration of measure” phenomenon means that, under soft conditions on the distribution of samples (uniform distribution is thus not required), the variance of any measure -distance, norm- remains fixed while its average increases with the dimension of the space. In particular, the standard deviation of the Euclidean norm of samples in a set remains almost constant when the dimension *d* of the space increases (for large dimensions). Naturally, the average Euclidean norm of the samples increases with the square root of the dimension *d*; the consequence of this is that, in large dimensions, the samples seem to be normalized (see [8] for a proof of this phenomenon in the case of the Euclidean norm). We will see in the next section that the “concentration of measure phenomenon” must be taken seriously into account when dealing with artificial neural networks.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

4. Local artificial neural networks

4.1. Local versus global models

Learning is the way to adapt parameters in a model, in function of known examples. *Learning* is the term used in the neural network community, while researchers in identification speak about *estimation*.

The literature often uses the term *local learning* for methods using combinations of local functions, as opposed to combinations of functions that span the whole input space. For example, RBFN (Radial-Basis Functions) networks use Gaussian functions which are considered as local since they rapidly vanish when the distance from their centres increases; RBFN networks are referred to as local. On the contrary, MLP (Multi-Layer Perceptrons) networks use sigmoid functions or hyperbolic tangents, which never vanish; MLP are referred to as global.

There is a tradition in the literature to consider that local models are less adapted to large-dimensional spaces than global ones. The reason for this is the empty space phenomenon. It has been shown intuitively in the previous section that the number of samples required for learning grows exponentially with the dimension of the space. This corresponds to our intuitive view of "filling" a space with local functions like Gaussian ones: if it is assumed that a Gaussian function corresponds to a fixed volume in the space, "filling" a distribution means to juxtapose a number of Gaussian functions proportional to the volume spanned by the distribution.

On the contrary, there is also a tradition in the literature to consider that global models do not have this limitation. Sigmoids (for example) span the whole input space, so it is assumed that a lower number of sigmoids is needed to fill the volume spanned by a distribution. We are convinced that this view of the problem is not correct; the following paragraphs explain three arguments in this direction.

First, it must be reminded that neural networks are interpolation tools, aimed to generalize information gathered on known data to other locations in the space. Interpolation means that we must have sufficient information in the surroundings of a point in order to interpolate at that point. Look for example at Figure 6. Both the plain and dashed lines are good approximators (interpolators) of the learning data (8 markers on the figure), while they are based on different assumptions (models) on the data. However, despite the fact that learning data range from $x = 0.5$ to $x = 1.7$, the plain and dashed lines give very different approximations around the value $x = 1$; in this case, we speak about extrapolation instead of interpolation. Of course, the example in Figure 6 is obvious, and should not even be commented. Nevertheless, we remind that our point is to show the difficulties of learning in high-dimensional spaces. To imagine how a distribution looks in high dimension, and in particular if the space is "filled" with data or not, is not obvious at all. Filling the space is related both to the density of points in the distribution, and to its convexity. In most situations, it is quite impossible to decide if we have to speak about interpolation or extrapolation.

Having this in mind, it is now clear that interpolation can be achieved only in regions of the space where there are "enough" data to interpolate; the words "richness of data" are sometimes used to define this concept. Even if the approximation function itself spans the whole space (for example in the case of sigmoids), interpolation has no sense in empty regions. The argument that sigmoids span a larger region than Gaussian functions is thus meaningless. Furthermore, the "lack of response" (more precisely outputs near to zero) of combinations of local functions in empty regions of the space could be used to appreciate the fact that there is not enough data to interpolate correctly.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

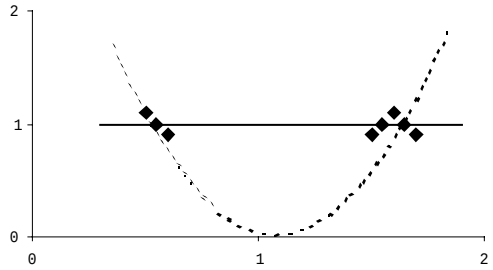

Figure 6: interpolation when the distribution of data is not convex. Plain and dashed lines are good interpolators of learning points, but extrapolate differently around $x = 1$.

The second argument is better explained by simple graphs. It must be reminded that neural network approximators, like MLP and RBFN, work by fitting a combination of sigmoid-like or Gaussian functions to the data to be approximated. Let us consider for simplicity that these combinations are linear. Figure 7 shows that the simple sum of two sigmoids looks like a Gaussian (weights, i.e. multiplying coefficients of the sigmoid in the sum, and thresholds have been chosen appropriately). In most situations when using a MLP, a phenomenon similar to this one will happen. Multiplying coefficients will adjust so that each region of the space is approximated by a weighted sum of a few basis functions (sigmoids). Naturally, since a sigmoid spans the whole space, the sigmoids used in the approximation in a region of the space will influence the approximation function in other regions. Nevertheless, another sigmoid easily cancels this influence, and so on.

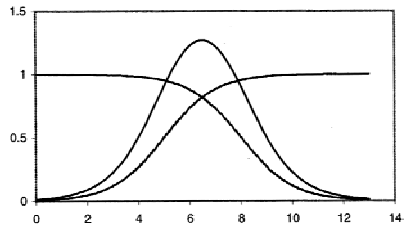

Figure 7: a weighted sum of sigmoids looks like a Gaussian function.

This phenomenon is common in MLPs, and is easily observed in small dimension with a limited number of sigmoids. The verification of this argument in large dimension and in a wide range of applications is a topic for further research.

The third argument is in the same range as the second one. It seems “natural” to think that approximation by a sum of local functions will lead to local functions having approximately identical widths (standard deviation in case of Gaussian functions), and positive weights (multiplying factors); let us just imagine how a set of Gaussian functions could be combined to fill a uniform distribution in a compact region of the space. However, except in some specific situations, it can be shown that this is not the case in practical applications. In particular, we can see that weights are often set to negative values, but also that widths are sometimes set to very large values (compared with the pairwise distance between adjacent kernels), making the contribution of this kernel to the approximation function very flat. Again, this argument reinforces the idea that local and global basis functions are not much different when they are combined to contribute to an approximation function in a high-dimensional space.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

4.2. What makes high-dimensional data learning with ANN difficult?

Artificial neural networks compute a function of their input vector; through the weights, this function is adapted to the problem to solve. Most of the functions involved in ANN models begin by computing some *distances* between the input vector and weights vectors. For example:

- Radial-Basis Function Networks (RBFN) compute sums of Gaussian functions whose argument is the *Euclidean norm* of the difference between the input vector and weight vectors;
- Vector quantization (VQ) methods and Kohonen self-organizing maps (SOM) also use the *Euclidean norm* of the difference between the input vector and so-called centroids (weight vectors) to select the winner;
- the first layer of Multi-Layer Perceptrons (MLP) takes the *scalar product* (another distance measure) between the input vector and weight vectors as argument to each non-linear function;
- etc.

It has been shown above that a consequence of the "concentration of measure" phenomenon is that the norm of all vectors seems to be constant in high-dimensional spaces. This result is therefore valid form the Euclidean norm of the difference between any two vectors, and also for the scalar product! Consequently, the input argument to the functions involved in neural networks becomes a constant, and does not depend anymore… from the inputs themselves! Of course, this argument is exaggerated; the consequence of the concentration of measure phenomenon is that all norms *concentrate* around their mean, not *are equal to*. Furthermore, we have no measure on *how much* they concentrate round their mean; we do not know from which dimension this phenomenon must be taken into account. Nevertheless, it is clear that for high-dimensional input vectors, the phenomenon must be taken into account, and appropriate measures taken in the learning itself. The following subsection gives two examples on how the functions involved in artificial neural networks could be modified to take this phenomenon into account.

4.3. Changing distance measures in neural networks

Simple changes in artificial neural network paradigms may help to reduce the problems related to high-dimensional vectors. Again, it is difficult to measure how much such changes help to overcome the problems; but experiences show that they help, sometimes considerably. Assessing the benefits, measuring them in an objective way, and suggesting other modifications in the same spirit is a topic for further research.

Let us take the example of Radial-Basis Function Networks (RBFN), defined by equation (4):

$$f(\boldsymbol{x}) = \sum_{j=1}^{P} w_j \, exp\left(-\frac{\left\| \boldsymbol{x} - \boldsymbol{c}_j \right\|^2}{h_j^2} \right), \tag{4}$$

where $\boldsymbol{x}$ is the input vector to the RBFN, $\boldsymbol{c}_j$ are to so-called centroids, h_j their widths, and w_j multiplying weights. Equation (4) defines a weighted sum of Gaussian functions centered on $\boldsymbol{c}_j$, with standard deviations proportional to h_j.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

The rationale behind the use of Gaussian functions is the fact that they are local, i.e. each function is fitted according to the learning points $\boldsymbol{x}$ that are *close* from $\boldsymbol{c}_j$, and the function vanishes for inputs *far* from $\boldsymbol{c}_j$.

It has been shown in the previous section that Gaussian functions in high-dimensional settings do not follow this intuitive view. In dimension as low as five, 75% of the surface beneath the Gaussian function is further away than the traditional 1.65 times standard deviation radius!

In order to overcome this problem (i.e. to make Gaussian functions in high-dimensional spaces *look like* scalar ones), it is suggested to replace the exponent 2 in equation (4) by a larger exponent g:

$$f(\boldsymbol{x}) = \sum_{j=1}^{P} w_j \, exp\left(-\frac{\left\| \boldsymbol{x} - \boldsymbol{c}_j \right\|^g}{h_j^g} \right). \tag{5}$$

The descending slope of the local function is now accentuated, so that a larger part of the surface beneath the function is concentrated near its center. The local functions used in (5) are called "super-Gaussian functions"; reference [6] shows an example of how parameter g can be evaluated, in the context of density estimation.

Another possibility to modify equation (4) is to replace the Euclidean norm by a higher-order one, defined by

$$norm_h\left(\boldsymbol{x} - \boldsymbol{c}_j\right) = \sqrt[h]{\sum_{i=1}^{d} \left| x_i - c_{ji} \right|^h} \; ; \tag{6}$$

$h = 2$ gives the standard Euclidean norm. It may be shown on examples that using $h > 2$ diminishes the importance of the "concentration of measure" phenomenon, one of the problems encountered with high-dimensional functions. Intuitively, one understands that larger h should be chosen for higher-dimensional spaces; however, it seems that there is still no theoretical result giving an estimation of the value of h that should be chosen in a specific situation; again this is a topic for further research.

The two suggestions above are only examples of what could be done in high-dimensional settings. Similarly, one could think to replace the scalar product between the input and the weight vectors by another distance measure in Multi-Layer Perceptrons, or to use a higher-order distance in the selection of the winner, in the context of Vector Quantization and Self-Organizing Maps.

5. Intrinsic dimension of data

5.1. Redundancy and intrinsic dimension

Despite all efforts that could be undertaken, learning high-dimensional data remains difficult. For this reason, each learning task should begin by an attempt to *reduce the dimension of the data*. In a previous section, it was argued that data in real problems often lie on or near submanifolds of the input space, because of the redundancy between variables. While redundancy if often a consequence of the lack of information about which type of input variable should be used, it is also helpful in the case where a large amount of noise is unavoidable on the data, coming for example from measures on physical phenomena. To be convinced of this positive remark, let us just imagine that the *same*

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

physical quantity is measured by 100 sensors, each of them adding independent Gaussian noise to the measurement; averaging the 100 measures will strongly decrease the influence of noise on the measure! The same concept applies if m quantities are measured by n sensors, with $n > m$.

Before thinking to reduce the dimension of the data, one should know how much the variables are redundant. Let us have a look to Figure 8, which shows the well-known "horseshoe" distribution in a 3-dimensional space. Looking to the set of data, we are convinced that we should project it on a 2-dimensional surface in order to reduce the dimension. Two is the number of *degrees of freedom* of the variables in the set, more commonly named the *intrinsic dimension* of the data.

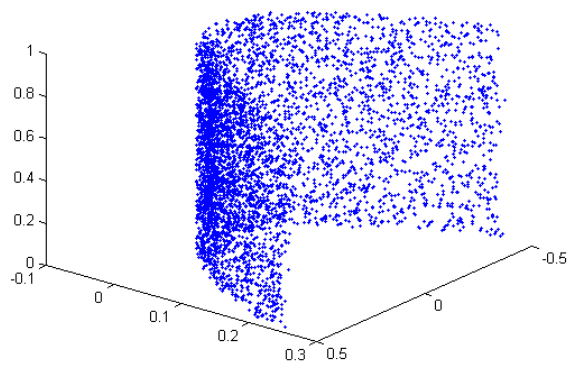

Figure 8. Horseshoe 3-dimensional distribution.

Estimating the intrinsic dimension of data may be a hard task. First, unlike the distribution in Figure 8, real distributions have varying intrinsic dimensions. The intrinsic dimension is thus a local concept, and any attempt to project the data should be made on a local basis too.

Secondly, the most used method in statistics to estimate the number of relevant variables is PCA (Principal Component Analysis); note that PCA is a projection method, but that the number of relevant variables may be found by looking to the eigenvalues of the covariance matrix of the data. Nevertheless, the PCA is a *linear* projection method; this means that PCA will perfectly work when the submanifold is a (hyper-)plane, but that it will fail in all other cases, including the example in Figure 8.

We advocate the use of two types of methods to find the intrinsic dimension of data.

5.2. Box counting methods

Let us recall that the volume of a d-dimensional cube is proportional to r^d, where r is edge of the cube. If we can make the hypothesis that the local density of points is constant over regions of "sufficient" size, the number of points contained in such a cube will be proportional to r^d if the data "fill" the space. However, if the intrinsic dimension of the data is m, with $m < d$, this number of points will be proportional to r^m. Imagine for example a 2-D surface in a 3-D space: any 3-D cube of increasing edge will contain a number of points growing with the square of the edge length, provided that the intersection of the cube with the distribution is approximately plane. A graph of the logarithm of the number of points contained in such a cube versus the logarithm of its edge length will thus have a slope equal to the local intrinsic dimensionality.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

The estimation of the local dimensionality at each sample of the input space obviously requires the computation of the mutual distance between any pair of points, which is computationally cumbersome when the dataset is large. Conversely, when the dataset is small, the condition of constant density over "sufficiently large" regions of the space is seldom achieved.

An easier method to implement is the so-called box-counting method [8], based on the following principle. The *d*-dimensional input space is divided into *d*-boxes of decreasing size. When the size of the box will be small compared to distance between neighbouring points in the dataset, the number of non-empty boxes will be proportional to r^{-m} where r is the length of an edge. For example, the number of boxes intersecting a 1-D string in a 3-D space is roughly equal to the length of the string divided by r, while the number of boxes intersecting a 2-D surface in a 3-D space is roughly equal to the area of the surface divided by r^2. A graph of the logarithm of the number of non-empty boxes versus the logarithm of the inverse of r will thus have a slope equal to the intrinsic dimensionality.

The Grassberger-Procaccia method [9] is similar to the box-counting procedure while different in its implementation. In the Grassberger-Procaccia method, all distances between any pair of points are computed. The method then leads to a global intrinsic dimensionality (instead of a local one). However, the advantage is that N points give $N(N-1)/2$ mutual distances, so that the number of points necessary to have an acceptable estimation of the intrinsic dimension is lower than in the box-counting method.

While the box-counting method is probably more convenient to implement than the Grassberger-Procaccia one, it suffers from two drawbacks in practice. First, when the intrinsic dimension is smaller than the dimension of the space (which is usually the case when the goal is to estimate the intrinsic dimension!), the number of empty boxes dramatically increases, leading to an inefficient computational load. Secondly, the local character of the intrinsic dimension is only preserved when the largest of the boxes considered in the method is small enough to keep the constant density hypothesis valid (this makes the first drawback even worse).

Figure 9 shows the box-counting method applied to the intrinsic dimension estimation of a 1-D string in a 2-D space (from [8]).

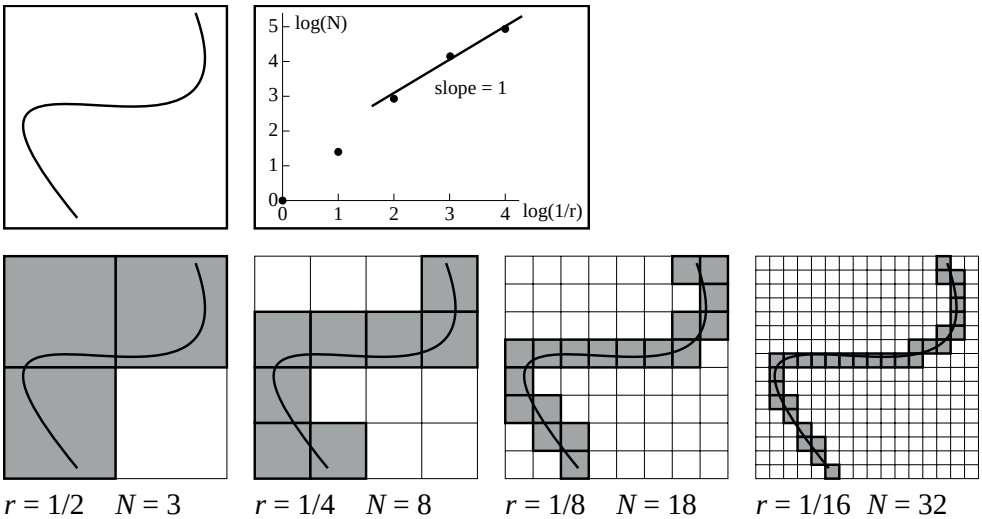

Figure 9: intrinsic dimension estimation of a 1-D string in a 2-D space by the box-counting method (from [8]).

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

5.3. A posteriori dimension estimation

We consider a second way to estimate the intrinsic dimension of a dataset. In most situations, the information about the intrinsic dimension is required for a further projection of the data space onto a smaller dimensional one. Contrary to PCA, nonlinear projection methods require determining the dimension of the projection space before computing the projection itself.

A simple idea is then to try the projection for several dimensions of the projection space, and to evaluate the results. Projections onto spaces of higher dimension than the intrinsic one will be "good", in the sense that the projection will be nearly bijective. Projections onto space of smaller dimension than the intrinsic one will be "bad", in the sense that points far from each other in the input space will be eventually projected to the same or similar locations in the projection space.

This method presents two major difficulties. First, it is clear that it will be computationally intensive, since the projection method itself must be considered for several dimensions of the projection space, rather than once. Secondly, an appropriate criterion must be defined to evaluate the quality of the projection; furthermore, even with an adequate criterion, the limit between "good" and "bad" projections may be difficult to set in practical situations.

Nevertheless, it has a strong advantage. As written above, the knowledge of the intrinsic dimension is often a prerequisite for a further projection of the data. Linking the measure of the intrinsic dimension to the projection makes sure that the right dimension measure is evaluated. Indeed, the reader noticed that we purposely did not give any clear definition of the concept of intrinsic dimension. The reason is that the definition is related to its measure, so that nothing prevents us to find different results when using the box-counting, Grassberger-Procaccia or any other method!

A further refinement of the *a posteriori* method is to remember that the non-linear projection itself serves as preprocessing to a learning task (function estimation, classification, etc.). Why not then simply try several dimensions for the non-linear projection, then perform the learning task on *each* result (with appropriate cross-validation methods), and finally select the dimension which gives the best result? In fact, this is the ultimate method, since it makes no assumption on any intermediate result (intrinsic dimension, projected distribution, etc.) but directly measures the result of the whole process. Nevertheless, of course, the computational load is dramatically increased. Despite its drawbacks, this last method should be considered in difficult situations, for example when the number of points is low, or when other measures of the intrinsic dimension do not give clear results.

6. Non-linear projection

When the fractal dimension m of the data is known (possibly as a local indicator rather than a global one), it should be possible to project the data from a d-dimensional space to a m-dimensional one: the data in Figure 9 could be projected from a 2-dimensional space to a 1-dimensional one, and those in Figure 8 from a 3-dimensional space to a 2-dimensional one. However, as the data in these figures and in most real situations are concentrated around a submanifold but *not* around a hyperplane, linear techniques like PCA (Principal Component Analysis) cannot be used. This is not good news; PCA is easy to use, gives straightforward and unique results, and does not suffer too much from numerical problems if some precautions are taken. Unfortunately, in our context, PCA is rarely sufficient. This

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

does not mean that PCA is not useful. When handling real data in high-dimensional spaces, it often happens that *some* variables are correlated to others. The use of PCA can be useful to make a first projection, and already reduce sometimes significantly the number of variables. For example, the data from Figure 8 could be the result of the projection to a 3-dimensional space of 4-dimensional data, where the fourth one was a linear combination of the three first ones. But projection to a 2-dimensional space is not possible by PCA in this case.

Non-linear projection is a hot topic. Despite the fact that some methods are known from many years now, the recent needs in data analysis techniques considerably raised the interest towards non-linear projection. A lot of tools or algorithms that can be used for nonlinear dimension reduction exist in the literature. It is not our intention here to make an exhaustive review of these techniques. Some are used in data analysis, some in statistics, some in the neural network community, or in other areas, and it is difficult to compare them objectively in general situations. We have to remind that, unlike linear methods such as PCA, nonlinear methods suffer from two difficulties concerning the evaluation of their performances:

1. nonlinear methods are usually adaptive and/or iterative; this means that parameters (strongly) influence the convergence of the algorithm. Making a method obtaining good results is thus often a question of efforts devoted to parameter tuning. Of course, a primary objective in designing the methods themselves is to make them as insensitive as possible to all parameters. Nevertheless, some sensitivity still remains in most cases.
2. Quality criterions are not so obvious as in linear cases. As described in a previous section, Euclidean distance measures may loose their meaning in high dimensions. Furthermore, most standard correlation criteria are based on second-order relationships; they are thus adapted to measure linear projections, but not non-linear ones.

In the following, we will briefly mention two types of methods used for dimension reduction: Kohonen maps on one side, and Multi-Dimensional Scaling – Sammon's mapping – Curvilinear Component Analysis on the other side. The first type is known in the neural network field, while the second one is mostly known is statistics.

6.1. Kohonen maps

Everybody working with neural networks knows Kohonen Self-Organizing Maps (SOM) [10] as an efficient clustering, quantization, classification and visualization tool; few authors however use it as a dimension reduction tool. Nevertheless the use of the SOM algorithm in that purpose is extremely simple, and may be efficient in some situations. Figure 10 shows a rectangular SOM after convergence on a horseshoe-shaped distribution similar to Figure 8.

The projection consists in using the 2-dimensional index of the centroids on the grid rather than their 3-dimensional coordinates. Of course, this projection is discrete (there is a finite number of centroids); simple (linear or polynomial) interpolation between adjacent centroids however makes this projection continuous.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

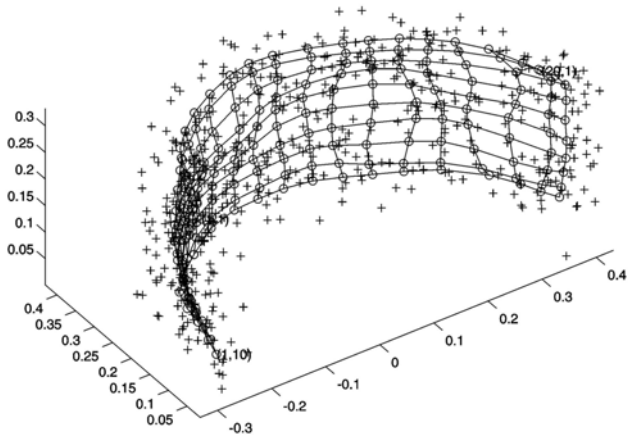

Figure 10: Horseshoe distribution and resulting Kohonen map. From [11].

Two comments must be done about the use of SOM as a non-linear projection method.

- Almost all applications of Kohonen maps in the literature use 2-dimensional grids; only a few ones use 1-dimensional strings, and even less use 3- or more-dimensional meshes. This is clearly a limitation, as it cannot be expected that real data will have an intrinsic dimension less than three… Nevertheless, on a theoretical point of view, nothing prevents us to use 3- or more-dimensional meshes; the convergence of the SOM algorithm faced to the empty space phenomenon in larger-dimensional spaces is however an unanswered question.
- The SOM algorithm is designed to preserve the *topology* between the input and grid spaces; in other words, to input points close one from another will be projected on the same or on close centroids. Nevertheless, the SOM algorithm does not preserve *distances*: there is no relation between the distance between two points in the input space and the distance between the corresponding centroids.

In theory, the last comment is not a limitation. Indeed, what we are looking for with (non-linear) projection, is that discrimination between different data remains possible: two different data in the input space must not be projected to the same location in the output space. In other words, the projection must be bijective. The topology preservation property of the SOM is a way to increase the smoothness of the projection (therefore to facilitate its design). But distance preservation is not strictly necessary!

6.2. Multi-dimensional scaling, Sammon's mapping and Curvilinear Component Analysis

Despite this last comment, many authors prefer to use tools designed to preserve the distances between the input and projection spaces. Our opinion is that the use of these methods is often preferred because of the traditional limitation of Kohonen maps to 2-dimensional grids, rather than the distance-preservation property.

Multi-dimensional scaling [12-13], Sammon's mapping [14] and Curvilinear Component Analysis [15] are methods based on the same principle: if we have n data points in a d-dimensional space, they try to place n points in the m-dimensional projection space, keeping the mutual distances between any pair of points unchanged between the input space and the corresponding pair in the projection space. Of course, having this condition strictly fulfilled is impossible in the generic case (there are $n(n-1)$ conditions to satisfy with nm degrees of freedom); the methods then weight the conditions so that those on shorter

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

distances must be satisfied more strictly than those on large distances. Weighting aims at conserving a local topology (locally, sets of input points will resemble sets of output points).

In the case of nonlinear Multi-Dimensional Scaling (MDS) [12-13], the objective function is simply the ratio of the input distances by the output distances, weighted in such a way that small output distances are more important than large ones.

Sammon's mapping [14] is similar to MDS. However, the objective function is now a mean square error of the differences between distances (between pairs of samples) in the input and output spaces. Contrary to MDS, weighting is done with respect to input distances in Sammon's mapping.

Curvilinear Component Analysis (CCA) [15] also measures the mean square error between distances. But contrary to Sammon's mapping, weighting is done with respect to distances in the projection space.

Dematines claims that this modification enhances the quality of the projection in many situations; our own experience with these algorithms confirms this claim. Another difference is that CCA does not use the n original points, but a smaller number k of centroids obtained after vector quantization of the input space. This decreases the amount of computations, but also facilitates the convergence of the algorithm because the quantization removes the noise on the data to some extend. Nevertheless, the CCA method still suffers from the difficulty to choose the crucial adaptation parameters of the algorithm, and from bad unfolding of difficult databases (for example those where some cutting is necessary to unfold the submanifold). [16] explains how some of these drawbacks can be avoided. Without going into the details of the choice of the adaptation parameters, the CCA method consists in the optimization of criterion

$$E = \sum_{i=1}^{k} \sum_{j=1}^{k} \left(X_{ij} - Y_{ij}\right)^2 F\left(Y_{ij}, \lambda\right), \tag{7}$$

where X_{ij} is the distance between points x_i and x_j in the input space, Y_{ij} the distance between the corresponding points y_i and y_j in the projection space, and F a decreasing function (with parameter λ) of the distances Y_{ij}. A simple step function with threshold λ (decreased during the convergence) may be used.

As the weighting function depends on the distances in the output spaces, a recursive procedure is needed to find the locations of the point y_i in the projection space. Conventional gradient descent on the error function E can be particularly tedious and computationally heavy; Demartines then suggests a simplified procedure where each point y_i is in turn considered as fixed, and all other points around y_i are moved. Under some supplementary assumptions, this lead to the adaptation rule

$$\Delta y_j = \alpha(t) F\left(Y_{ij}, \lambda\right)\left(X_{ij} - Y_{ij}\right) \frac{y_j - y_i}{Y_{ij}}, \quad \forall j \neq i. \tag{8}$$

In order to facilitate the convergence on difficult databases, we suggested in [16] using a kind of "curvilinear distance" instead of the standard Euclidean distance for X_{ij}. This curvilinear distance is measured as the shortest path via links drawn between adjacent centroids. Using this distance increases the possibility for the method to respect criterion (7), and therefore facilitates the convergence of the algorithm. As an example, the distance between any two centroids in Figure 10 is no more the Euclidean distance, but rather the sum of the lengths of all segments drawn in the figure that are necessary to join the two centroids (shortest path).

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

Figure 11 shows the result of the projection of a sphere on a 2-dimensional space, using the CCA algorithm with curvilinear distances.

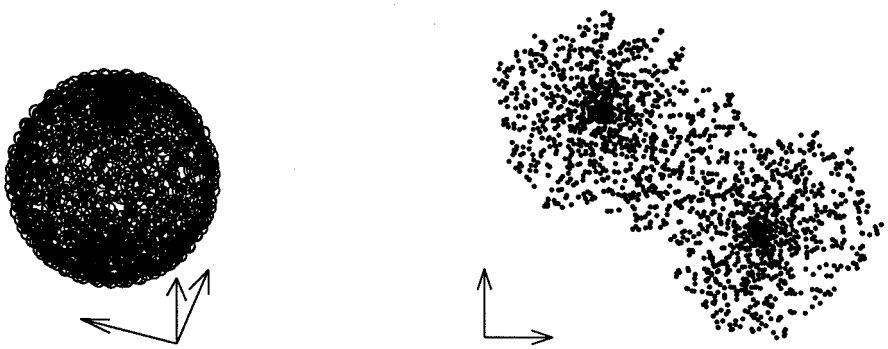

Figure 11: Left: sphere in a 3-dimensional space. Right: its projection on a 2-dimensional plane.

7. An example of dimension reduction

As an illustrative example, we consider the forecasting of the Belgian BEL20 stock market index. The application of time series forecasting to financial market data is a real challenge. The efficient market hypothesis (EMH) remains up to now the most generally admitted one in the academic community, while essentially challenged by the practitioners. Under EMH, one of the classical econometric tools used to model the behaviour of stock market prices is the geometric Brownian motion. If it does represent the true generating process of stock returns, the best prediction that we can obtain of the future value is the actual one (they follow a random walk). Results presented in this section must therefore be analysed with a lot of caution. Analyzing a rather local index as the BEL20 is however facilitated by the fact that local indexes are largely influenced by the evolution of larger ones after a small delay (some hours, one day). As the values of these larger indexes are know, they can be incorporated in the model used to forecast the BEL20. For these exogenous variables, we selected international indices of security prices (SBF 250, S&P500, Topix, FTSE100, etc), exchange rates (Dollar/Mark, Dollar/Yen, etc), and interest rates (T-Bills 3 months, US Treasury Constant Maturity 10 years, etc). We used 42 technical indicators in total [17] (chosen according to [18] and [19]), based on these exogenous variables and of course also on the past values of the series.

We used 2600 daily data of the BEL20 index over 10 years to have a significant data set. The problem considered here is to forecast the sign of the variation of the BEL20 index at time t+5, from available data at time t.

If we carry out a Principal Component Analysis (PCA) on these 42 variables, we note that 95% of the original variance is kept with the first 25 principal components: 17 variables can be removed without significant loss of information. The PCA is used to facilitate the subsequent processing by the CCA algorithm (lower computational load and better convergence properties).

The time series of the target variable, x_{t+5}, whose sign has to be predicted, is illustrated in Figure 12.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

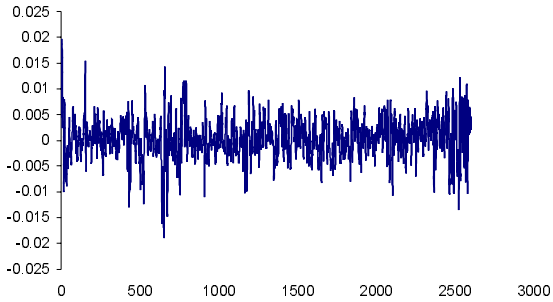

Figure12: Daily values of the BEL20 index.

This variable has to be predicted using the resulting 25 variables selected after PCA. The interpolator we use is a conventional Radial-Basis Function (RBFN) network (RBFN). Our interest goes to the sign of the prediction only, which will be compared to the real sign of the target variable.

The Grassberger-Proccacia method is used to estimate the intrinsic dimension *m* of the data set; we obtain an approximate value of 9. We then use the CCA algorithm to project the 25-dimensional data (after PCA) on a 9-dimensional space. The RBFN interpolator is used on the resulting 9-dimensional input vectors.

The network is trained with a moving window of 500 data. Each of these data consists in a 9-dimensional input vector (see above) and a scalar target (variation of the BEL20 index). We use 500 data as a compromise between

- a small stationary set but insufficient for a successful training, and
- a large but less stationary training set.

For each window, the 500 input-target pairs form the training set, while the test set consists in the input-target pair right after the training set. This procedure is repeated for 2100 moving windows. On average, we obtain 60,3% correct approximations of the sign of the series on the training sets, and 57.2% on the test sets. These results are encouraging and are far above what can be obtained with RBFN or other interpolators trained on the initial 42-dimensional vectors.

Moreover, it can be seen that better results are obtained during some periods and worse results during others. The first ones correspond to time periods where the series is more stationary than the last ones. Figure 13 represents a moving average on 90 days on the results of the prediction. It clearly shows that that the prediction results themselves do not form a random series: when the forecasting is correct over several consecutive days, the probability that it will be correct at the next time step is high.

To quantify this idea, we filter the results with the following heuristics. We look at the average of sign predictions (correct – not correct) over the last 5 days. If this average increases or remains constant at time *t*, then we take the forecasting at time *t*+1 into consideration. If it decreases, then we disregard the forecasting at time *t*+1. With this method, we keep 75.4% of the forecasts; the average score of correct prediction rises to 65.3% (about 70% of increases and 60% of decreases). This way of working is a first attempt to use such mathematical procedure in a real-world financial context.

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

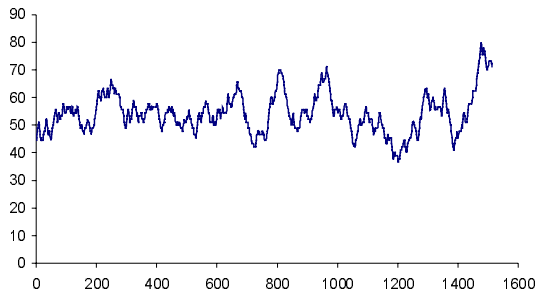

Figure 13: Percentage of correct approximations of the sign on a 90-days moving window.

8. Conclusion

Artificial neural networks are learning methods used in complex problems, where it is difficult to find appropriate physical models, or where most other modeling techniques fail because of their linearity, polynomial character, etc. Most of these complex, real problems, involve observations made of several (many) variables. The observations are thus points in high-dimensional spaces, and unfortunately most adaptive techniques and neural network algorithms are not designed to take this high dimensionality into account. However, the same techniques are effectively used with high-dimension data, and prove to be successful in some cases. But they fail in other situations, because of the "curse of dimensionality" and the "empty space phenomenon".

This paper emphasizes on the problem that can be encountered when high-dimensional data are handled in the same way as low-dimensional ones. It shows that some basic concepts as the interpolation-extrapolation dilemma, the use of Euclidean distances, the local character of Gaussian functions, etc. are misleading when used with high-dimensional data.

We then show through a few examples that even local artificial neural networks can be modified easily to take the curse of dimensionality effect into account. We argue that the difficulties related to the use of high-dimensional data are equivalent with local (RBFN, SOM, etc.) and with global (MLP, etc.) neural networks.

Finally, it is shown how the dimension of data can be reduced, with non-linear techniques, in order to improve the results of a subsequent learning task (classification, approximation, forecasting, etc.), with algorithms like the "Curvilinear Component Analysis". An application of this technique to the forecasting of financial time series concludes the discussion.

Acknowledgements

Michel Verleysen is a Senior Research Associate of the Belgian FNRS (National Fund for Scientific Research).

Limitations and Future Trends in Neural Computation, S. Ablameyko et al. (Eds.),
IOS Press, 2003, pp. 141-162

References

[1] D. L. Donoho, High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality. Lecture on August 8, 2000, to the American Mathematical Society "Math Challenges of the 21st Century". Available from http://www-stat.stanford.edu/~donoho/.

[2] M. Turk, A. Pentland, Eigenfaces for Recognition, *J. Cognitive Neuroscience*, **3**-1 (1991) 71-96.

[3] R. Bellmann, Adaptive Control Processes: A Guided Tour. Princeton University Press, 1961.

[4] A. Barron, Universal Approximation Bounds for Superpositions of a Sigmoidal Function, *IEEE Tr. on Information Theory*, **8**-3 (1993) 930-945.

[5] D. W. Scott, J. R. Thompson, Probability density estimation in higher dimensions. In: J.E. Gentle (ed.), Computer Science and Statistics: Proceedings of the Fifteenth Symposium on the Interface, Amsterdam, New York, Oxford, North Holland-Elsevier Science Publishers, 1983, pp. 173-179.

[6] P. Comon, J.-L. Voz, M. Verleysen, Estimation of performance bounds in supervised classification, *European Symposium on Artificial Neural Networks*, Brussels (Belgium), April 1994, pp. 37-42.

[7] B.W. Silverman, Density estimation for statistics and data analysis. Chapman and Hall, 1986.

[8] P. Demartines, Analyse de données par réseaux de neurones auto-organisés. Ph.D. dissertation (in French), Institut National Polytechnique de Grenoble (France), 1994.

[9] P. Grassberger, I. Procaccia, Measuring the strangeness of strange attractors, *Physica D*, **56** (1983) 189-208.

[10] T. Kohonen, Self-Organizing Maps. Springer Series in Information Sciences, vol. 30, Springer (Berlin), 1995.

[11] A. Choppin, Unsupervised classification of high dimensional data by means of self-organizing neural networks. M.Sc. thesis, Université catholique de Louvain (Belgium), Computer Science Dept., June 1998.

[12] R. N. Shepard, The analysis of proximities: Multidimensional scaling with an unknown distance function, parts I and II, *Psychometrika*, **27** (1962) 125-140 and 219-246.

[13] R.N. Shepard, J.D. Carroll, Parametric representation of nonlinear data structures, *International Symposium on Multivariate Analysis*, P. R. Krishnaiah (ed.) pp. 561-592, Academic Press, 1965.

[14] J.W. Sammon, A nonlinear mapping algorithm for data structure analysis, *IEEE Trans. on Computers*, **C-18** (1969) 401-409.

[15] P. Demartines, J. Hérault, Curvilinear Component Analysis: a self-organizing neural network for nonlinear mapping of data sets, *IEEE Trans. on Neural Networks*,. **8**-1 (1997) 148-154.

[16] J. Lee, A. Lendasse, N. Donckers, M. Verleysen, A robust nonlinear projection method, ESANN'2000 (European Symposium on Artificial Neural Networks), Bruges (Belgium), April 2000, pp. 13-20, D-Facto publications (Brussels).

[17] A. Lendasse, J. Lee, E. de Bodt, V. Wertz, M. Verleysen, Input data reduction for the prediction of financial time series, ESANN'2001 (European Symposium on Artificial Neural Networks), Bruges (Belgium), April 2001, pp. 237-244, D-Facto publications (Brussels).

[18] A.N. Refenes, A.N. Burgess, Y. Bentz, Neural networks in financial engineering: a study in methodology, *IEEE Transactions on Neural Networks*, **8**-6 (1997) 1222-1267.

[19] A.N. Burgess, Nonlinear model identification and statistical significance tests and their application in financial modeling. In *Artificial Neural Networks*, Proceedings of the Inst. Elect. Eng. Conf., 1995.