Contributed article

Two methods for encoding clusters

Pierre Courrieu*

Laboratoire de Psychologie Cognitive, UMR CNRS 6561, Université de Provence, 29 avenue Robert Schuman, 13621 Aix-en-Provence cedex 1, France

Received 17 February 2000; accepted 28 October 2000

Abstract

This paper presents two methods for generating numerical codes representing clusters of $\mathbb{R}^n$, while preserving various topological properties of data spaces. This is useful for networks whose input, or eventually output, consists of unordered sets of points. The first method is the best one from a theoretical point of view, while the second one is more usable for large clusters in practice.

Keywords: Cluster codes; Neural network input and output representations; Pseudo-sequences; Unordered sets; Encoding problem

1. Introduction

There are applications in which the input, or eventually the output, of a neural network is not a point but a cluster, that is an unordered, countable finite set of points of $\mathbb{R}^n$. A frequent difficulty is to find an appropriate representation for such an input, given that there is no natural order of points in $\mathbb{R}^n$ for $n > 1$. Any permutation of a cluster’s points is a priori equivalent to other ones, but distinct permutations provide distinct input vectors, and learning the equivalence of permutations is a very complex problem given that for a set of $m$ points there are $m!$ possible permutations. One can always build an artificial order of points in $\mathbb{R}^n$, for example using a Peano–Hilbert scanning. Unfortunately, this type of procedure never allows for preserving the topology of data spaces, and a small variation of data points can lead to a large variation in the representation. This results in major difficulties for learning regular functions on the space of such representations. A quite similar problem occurs with pseudo-sequences of points, that is when data points are ordered with respect to a natural variable, such as a time coordinate, but the underlying process (generating data points) is not actually sequential or is a random mixture of several sequences. In this case, small and possibly random variations of time coordinates can modify the order of points, resulting in large variations of the input representation.

The problem we address here is: find a mapping from the set of clusters of $\mathbb{R}^n$ to a set of real vectors (or matrices), referred to as ‘cluster codes’, such that (1) any cluster has a unique code, (2) distinct clusters have distinct codes, and (3) to any continuous movement of points in a cluster corresponds a continuous variation of code components. Such a mapping would be appropriate for encoding input clusters. Now, if the application requires that one can decode output cluster codes, we also need that (4) the mapping is invertible (which implies (2)). A mapping with the four above properties is in general an homeomorphism between the data space and the code space. However, one must take care that the code space can be a special part of $\mathbb{R}^k$, which is not as simple as $\mathbb{R}^k$ itself. To date, we know of no solution which can be applied to all practical problems. However, there are various solutions for various families of problems. Two of these solutions are presented hereafter. Note that the problem addressed here is to find a finite exact representation of data, in a format which ensures that representations of various clusters can be compared. This is a problem different (and much less studied in the literature) from that of approximating data distribution by a density of probability (Husmeier & Taylor, 1998; Specht, 1990; Traven, 1991) or an attractor (Barnsley, 1993; Diaconis & Freedman, 1999). The problem is in some way related to the ‘encoding problem’ in neural self associators (Rumelhart, Hinton & Williams, 1986). However, such encoders require prior learning, and they cannot guarantee property (1), since a permutation of a cluster’s points generally results in a change of the internal representation.

2. First method: polynomial encoding of clusters

For encoding a cluster in $\mathbb{R}$ or $\mathbb{R}^2$ only, one can consider a real or complex polynomial of the form $P(z) = \prod_{j=1}^m (z - z_j)$,
which has exactly $m$ real or complex roots, depending on the dimension. These roots are obviously the $z_i$'s, which are the coordinates of the cluster’s points. The product in real or complex algebra is commutative and associative, then $P(z)$ does not depend on the order in which the $m$ roots are taken into account. As a consequence, $P(z)$ is uniquely associated to the unordered set of roots (that is the cluster to be encoded). One can expand the polynomial $P(z)$, this expansion resulting in a sequence of $(m + 1)$ real or complex coefficients which is a possible cluster code satisfying all requirements (1)–(4). Unfortunately, as a consequence of the well known Frobenius theorem, one knows that there is no commutative and associative algebra in dimension larger than 2. There is an associative algebra in dimension 4 (quaternions algebra), but this algebra is not commutative. Then, the above sketched method cannot be extended for encoding clusters in dimension larger than 2.

The method described hereafter satisfies requirements (1)–(4) for any cluster of $R^n$, and for any $n$. So, it is theoretically a very general method, and it can be useful for theoretical purpose. However, there are restrictions in practice due to the fact that the generated code size increases rapidly with the cluster dimension ($n$) and the cluster size ($m$). Moreover, decoding is a complex operation, and when the code is only an approximation the result of decoding can be very approximate. Therefore, the author recommends the practical use of this method only for encoding small clusters in low dimension spaces, and when the application does not require decoding. A possible application field is the encoding of game configurations, and the method is well adapted for encoding configurations which contain several distinct clusters of various sizes and dimensions.

2.1. Data projection on the half unit sphere of $R^{n+1}$

Let $\{X_j \in R^n; 1 \leq j \leq m\}$ be the set of points of a cluster, where one assumes that no point has infinite coordinates. Choose an origin point $O \in R^n$ and a real $r > 0$ such that, for any $j$, $\|X_j - O\| \leq r$, where $\|$ denotes the Euclidean norm. Then the projection on the half unit sphere of $R^{n+1}$ of the point $X_j = (x_{j1}, \ldots, x_{jn})$ is given by:

$$u_{ij} = (x_{ij} - o_{ij})/r, \quad 1 \leq i \leq n,$$

and

$$u_{0j} = (1 - \sum_{i=1}^{n} u_{ij}^2)^{1/2}.$$

The vector $U_j = (u_{0j}, u_{1j}, \ldots, u_{nj})$ is such that $\|U_j\| = 1$, and the projection is obviously reversible by $x_{ij} = ru_{ij} + o_i$, $1 \leq i \leq n$.

One can choose standard $O$ and $r$, and then project any cluster which is inside the sphere of $R^n$ of center $O$ and radius $r$. This sphere will be referred as the “encoding sphere”, and one can remark that projected coordinates are always continuous real functions of original coordinates inside the encoding sphere. One can also take for $O$ the center of gravity of the cluster, which provides translation invariance, and/or take $r = s \max_{1 \leq j \leq m} \|X_j - O\|$, $s \geq 1$, which provides scale invariance. In the latter case, any cluster of $R^n$ is inside its own encoding sphere.

The main interest of the projection on the half unit sphere of $R^{n+1}$ is that all vectors $U$ have unit norm, and then:

$$\frac{1}{2} \|U - U_j\|^2 = 1 - U_j \cdot U_j \quad \text{and} \quad 1 - U_j \cdot U_j = 0 \iff U + U_j \iff X = X_j.$$

2.2. Polynomial associated with a cluster

Consider the polynomial defined by:

$$P(U) = \prod_{j=1}^m (1 - U_j U_j')$$

where the $U_j$’s are the projections of the cluster’s points on the half unit sphere of $R^{n+1}$, $U$ is the projection of the variable, and $\cdot$ denotes the inner (dot) product of vectors.

This is a real polynomial of degree $m$ with $n + 1$ variables, and after Section 2.1, the set of roots of this polynomial is exactly the set of projections of the $m$ cluster’s points on the half unit sphere of $R^{2m+1}$. Given that the (real) product is commutative and associative, this polynomial does not depend on the order in which data points are taken into account. Since the projection on the half unit sphere is reversible, $P(U)$ is uniquely associated with the cluster in a given encoding sphere. Then, expanding $P(U)$, one obtains a sequence of coefficients which is a possible cluster code. The expansion of this algebraic polynomial is finite, and it is of the form:

$$P(U) = \sum_{k=0}^{m} \sum_{a_0 + a_1 + \cdots + a_n = k} (-1)^k c[a_0, a_1, \ldots, a_n] u_{00}^{a_0} u_{1j}^{a_1} \cdots u_{nj}^{a_n},$$

where the $a_i$’s are integer exponents of the (projected) variable’s coordinates. The real coefficients $c[\cdot]$ are the code components, and they can be described in the following way. Let $P[m, k]$ denote the set of parts with $k$ elements of the integer set $\{1, 2, \ldots, m\}$, and let $Q[a_0, a_1, \ldots, a_n]$ denote the set of all distinct permutations of the set which contains $a_0$ integer values 0, $a_1$ integer values 1, \ldots, $a_n$ integer values $n$. With $k = a_0 + a_1 + \cdots + a_n$, one has:

$$c[0,0,\ldots,0] = 1,$$

$$c[a_0, a_1, \ldots, a_n] = \sum_{\{j_1, \ldots, j_k\} \in P[m,k]} \sum_{\{a_{0j_1}, \ldots, a_{nj_k}\} \in Q[a_0, a_1, \ldots, a_n]} \prod_{s=1}^k u_{i,j_s}.$$
Example. — With \( n = 2 \) and \( m = 3 \), one must expand the polynomial:
\[
P(u_0, u_1, u_2) = (1 - u_0)u_0 - u_1u_1 - u_2u_2 - (1 - u_0)u_0
\]

The expansion by hand is quite tedious but easy, and one obtains:
\[
P(u_0, u_1, u_2) = 1 - (u_0 + u_0u_0 + u_0u_0u_0)u_0 - (u_1 + u_1u_1
\]

Fortunately, there is another way of computing the coefficients of the expansion, as we shall see in the next section.

### 2.3. Encoding procedure

#### 2.3.1. Computing the coefficients

The following procedure provides the same coefficients than those of Section 2.2, however, its recurrent form makes it much easier to implement on standard computers. The encoding procedure is presented in pseudo-Pascal notation:

\[
c[0, 0, ..., 0] = 1
\]

for \( j = 1 \) to \( m \) do

for \( k = j \) downto 1 do

for \( a_0; = 0 \) to \( k \) do

for \( a_1; = 0 \) to \( (k - a_0) \) do

\[
\ldots
\]

for \( a_{n-1}; = 0 \) to \( (k - \sum_{i=0}^{n-2} a_i) \) do

begin

\[
a_c; = k - \sum_{i=0}^{k-1} a_i;
\]

if \( k = j \) then \( c[a_0, a_1, ..., a_n] := 0; \)

\[
c[a_0, a_1, ..., a_n] := c[a_0, a_1, ..., a_n] + \sum_{i=0}^{n>0} a_i c[a_0, ..., a_i - 1, ..., a_n];
\]

end;

Note. For readers unfamiliar with pseudo-Pascal notation, one can say that this is simply English mixed with mathematical notations. The semicolon is the statement separator, while ‘\( x' = y' \)’ means that \( x \) takes the value of the expression \( y \). The sequence of statements between ‘begin’ and ‘end’ must be repeated in each iteration governed by the ‘for’ statements.

#### 2.3.2. Addressing the coefficients in a vector

For clarity in the procedure (2.3.1) statement, we used a multidimensional array notation for \( c[\cdot] \), where we have in fact to obtain a vector. Moreover, the size of this multidimensional array would be of \( (m + 1)^{n+1} \) real numbers, while the total number of coefficients of the expansion is only \( K_{(n+2)}^m \). The combinatorial \( K_i^j \) can be recursively computed using the relations:

\[
K_i^j = K_i^j + K_{i-1}^{j-1}\]

Using this combinatorial, one can compactly address the coefficients in a one dimension array as follows:

\[
k := a_0 + a_1 + \cdots + a_n;
\]

\[
\text{index}(a_0, a_1, ..., a_n) := (k > 0)K_{(n+2)}^{k-1} + \]

\[
\sum_{j=0}^{a_0-1} K_n^{k-j} + (a_1 > 0) \sum_{j=0}^{a_1-1} K_{n-1}^{k-a_0-j} + \]

\[
\sum_{j=0}^{a_2-1} K_{n-2}^{k-a_0-a_1-j} + \cdots + \]

\[
\sum_{j=0}^{a_{n-1}-1} K_1^{k-a_0-\cdots-a_{n-2}-j},
\]

where we note that \( \{\text{false}\} = 0 \), and \( \{\text{true}\} = 1 \).

In practical implementations, the multi-index \([a_0, a_1, ..., a_n]\) which appears in Section 2.3.1 must be replaced by the scalar integer function \( \text{index}(a_0, a_1, ..., a_n) \) defined above. Indexed this way, the coefficients are ordered.
in increasing order of the total degree of the terms they weight. Then the total degree corresponding to the last non zero coefficient provides the number of points of the cluster. If the index of this coefficient is larger than $K_{n+2}^{k-1}$ but no larger than $K_n^k$ then the degree is $k$, and $m = k$. One can theoretically use an infinite vector for encoding any cluster of $R^n$, the unused components being set to zero.

2.4. Decoding

Properties (1) and (2) of the Introduction imply that there is a bijection between the set of clusters of $R^n$, inside a given encoding sphere, and the set of exact cluster codes. This guarantees that any exact cluster code is, in some way, decodable with an exact result. Now, in practice, codes to be decoded are in general approximations which are not necessarily exact cluster codes, since the set of exact cluster codes is only a subset of the set of real vectors. We sketch hereafter a method which theoretically allows for approximately decoding a real vector into a cluster. Of course, an exact result is accessible if the vector is an exact cluster code.

First one can find the number of points ($m$) from the last non zero coefficient of the vector $V$ to be decoded (see Section 2.3.2). Let $Q$ be the set of clusters of size $m$ in a given encoding sphere, and consider a cluster $q \in Q$, and denote $C(q)$ the cluster code associated with $q$ by the encoding method (Sections 2.1 and 2.3). Then a solution, denoted $q^*$, to the decoding problem is provided by solving the following global optimization problem:

$$q^* = \arg \min_{q \in Q} \| C(q) - V \|.$$

Since the search domain is bounded (encoding sphere) and the functional to be minimized is continuous (as a consequence of property (3)), there are global optimization algorithms whose convergence to a global minimizer is guaranteed (Courrieu, 1997; Theorem 1; Ingber & Rosen, 1992; Solis & Wets, 1981). This theoretically guarantees the solvability of the decoding problem. However, it was observed in practice that the solving time increases very fast with the dimension ($n$) and the size ($m$) of clusters. So, the above described decoding method cannot reasonably be recommended as a practical one, except for very small problems.

2.5. Computational test

We know that the encoding preserves basic topological properties of the data space. The object of the computational test is to see if metric relations are also preserved, and if they correctly reflect variations in data generating processes. For the computational test, we generated 20 uniformly random clusters of $m = 12$ points in $[-1, 1]^4$ (i.e. $n = 4$). Ten modified versions of each cluster were generated by adding a random quantity, uniformly sampled in $[-6, 6]$, to each coordinate of each point. The modification scale $\delta$ was experimentally varied from 0.0001 to 1.28, and each added quantity was eventually resampled until the modified coordinate was in $[-1, 1]$. The distance between the code $C$ of a cluster and the code $C'$ of a modified version was defined as the Euclidean distance of codes: $d(C, C') = \| C - C' \|$. We also computed, for comparison, the Hausdorff distance (associated to the Euclidean distance in $R^n$) between clusters in the data space (see Barnsley, 1993). Let $q$ and $q'$ be the two clusters to be compared, then their Hausdorff distance is defined by:

$$h(q, q') = \max(\max_{X \in q} \min_{Y \in q'} \| X - Y \|, \max_{Y \in q'} \min_{X \in q} \| Y - X \|).$$

Mean distances (with standard deviations on 19 degrees of freedom) are reported in Table 1.

As one can see in Table 1, the distance of codes is, on the average, a monotone increasing function of the modification scale, and is approximately proportional to the Hausdorff distance ($d(C, C') = 700$ Hausdorff). Hence, it seems that the encoding method generates a code space whose metric properties are compatible with those of the data space.

Out of curiosity, the decoding problem was tested on exact codes of very small clusters, $1 \leq m \leq 4$, in $[-1, 1]^4$. Decoding problems were exactly solved using the so called ‘Hyperbell algorithm’ (Courrieu, 1997). It turned out that the number of steps of the algorithm for finding exact solutions was about $455 \times 10^6$, where each step required encoding a cluster. Clearly, this is not a practical solution.

2.6. An example of application

The above described encoding method was recently used in sport’s science area for encoding and comparing basketball play configurations (Courrieu, Ripoll, Ripoll, Baratgin & Laurent, submitted; Baratgin, Ripoll, Ripoll, Courrieu & Laurent, submitted). Basketball play configurations can be schematized as illustrated in Fig. 1. In this type of representation, ground marks provide a reference, the dot stands for the ball, crosses represent attackers, and segments represent
defenders with their orientation. Coaches commonly use such representations for training basketball teams. A play configuration can be described as a set of three clusters: the ball which is a cluster of one point in the plane, the attacker team, which is a cluster of five points in the plane, and the defender team, which is a cluster of five points in a four dimension space since each defender is represented by his position in the plane (two coordinates) and his orientation (two more coordinates). A convenient way of representing a defender consists of taking his “left hand” and “right hand” plane coordinates. Then, it suffices to encode each of the three clusters as described above, and to concatenate the resulting codes, in fixed order, in a global vector (312 real coefficients). This vector unequivocally represents the multicluster play configuration.

Fig. 1 shows four close configurations (numbered from 1 to 4), each of them being obtained from the previous one by changing the position of one player, while the fifth configuration (X) is very different from the other ones. Euclidean distances between codes associated to these configurations were computed and they are shown at the bottom-right of Fig. 1. As one can see, these distances reflect the differences between play configurations in a quite natural way. More advanced results on this topic are reported in Courrieu et al. (submitted) and Baratgin et al. (submitted).

3. Second method: cluster codes depending on a separating variable

This method satisfies requirements (1) and (3) for any cluster, and requirements (2) and (4) in most cases, but not all. Its advantages are that the generated code is concise (nm real coefficients), and that decoding, when possible, is very simple. Moreover, a simple inspection of a code vector immediately shows whether or not decoding is allowed. This method is particularly adapted for encoding and decoding pseudo-sequences of points. A possible application field is the encoding of sets of seismic events, such events being rarely simultaneous if the time is measured with relatively high precision. Another possible application field is the
analysis of sequences of regularly sampled physical measures (e.g. acoustical or electrophysiological measures).

3.1. The one variable case

The one variable case is very simple since points of $\mathbb{R}$ are naturally ordered. Then it is sufficient to order data points in increasing order (or in fact any fixed order) of their values to obtain a cluster code which trivially satisfies requirements (1)–(4) in all cases. In particular, decoding is simply the identity mapping. In the case where all points of a cluster of $\mathbb{R}$ have distinct coordinates, we say that the variable satisfies requirements (2) and (4) if and only if the $t$ variable separates the cluster’s points. After Section 3.1, the set of clusters for which $t$ is not separating is a set of zero measure, however it is not an empty set. Now, a simple inspection of the column $C_1$ of the code immediately shows whether it contains equal coefficients, that is, if decoding is allowed or not. When a code was obtained as the output of an approximation process (neural network), it can happen that the components of $C_1$ are not in increasing order. This is not problematic, since the set of points provided by decoding will be the same, in permuted order (since $(Sp)^{-1} C_1 = p'S^{-1} C_1 = p' Y_1$).

However, if comparisons between codes are required, then it is necessary to reorder $C_1$ (and only $C_1$!).

The code has another property which can be useful in certain applications. Let $V$ be the matrix whose columns are the $V_j$’s, let $W$ be the matrix whose columns are the $C_j$’s, $2 \leq i \leq n$, and let $B$ be any $(n-1) \times (n-1)$ real matrix. Then, the cluster whose coordinate matrix is $(T, VB)$ has the code matrix $(T, WB)$, since the matrix $S$ only depends on $T$, and $W = SV$ implies that $SVB = WB$.

In other words, applying a linear transform to the data coordinates, without changing the first coordinate, results in applying the same transform to the code.

3.2. Encoding clusters of $\mathbb{R}^n$

Let $t$ be the first coordinate of $\mathbb{R}^n$, and consider a cluster of $m$ points $\{X_1 = (t_1, x_{i_1}, \ldots, x_{i_m}); 1 \leq j \leq m \}$ in $\mathbb{R}^n$, where points have been ordered in such a way that $t_j \leq t_{j+1}$, $1 \leq j \leq (m-1)$. Let $g(t)$ be a strictly monotonic continuous function, and form the square matrix $S = (sij)$, $1 \leq i \leq m$, $1 \leq j \leq m$, where $s_{ij} = [g(t_j)]^{i-1}$. All coefficients in the first row of the matrix $S$ have value $[g(t_1)]^0 = 1$, and remaining rows are successive integer powers of the second row coefficients. A matrix with such a structure has a determinant, called ‘Vandemonde determinant’, which is:

$$\det S = \prod_{1 \leq j \leq k \leq m} (s_{2k} - s_{2j}).$$

It is clear that this determinant is different from zero if and only if all coefficients in the second row have different values. Given that $g$ is strictly monotonic, one obtains:

$$\det S \neq 0 \iff t_j \neq t_k, 1 \leq j < k \leq m.$$

Now, consider the vectors $T = (t_j), 1 \leq j \leq m$, and $V_i = (x_{i_j}), 1 \leq j \leq m, 1 \leq i \leq n$. The $m \times n$ matrix $C$ whose first column is $C_1 = T$, and whose $i$th column is $C_i = SV_i$, $2 \leq i \leq n$, is a code which satisfies requirements (1) and (3) in all cases. The first column encodes only one variable ($t$), then after Section 3.1, the conventional increasing order of $C_1$ components is appropriate. On the other hand, the columns $C_i, i > 1$, do not depend on any ordering of points: let $p$ be an $m \times m$ permutation matrix, then a permutation of the points gives $C_i = Sp^t V_i = SV_i$, where $p^t$ denotes the transposed of $p$, which is equal to its inverse given that any permutation matrix is orthogonal. This guarantees the property (1), while the property (3) results from the fact that $g$ is continuous and the encoding only implies products and sums. Moreover, if $\det S \neq 0$, then $S$ is invertible and $V_i = S^{-1} C_i, 2 \leq i \leq n$, where $S$, and then $S^{-1}$, can be computed from the column $C_1$. In other words, this encoding satisfies requirements (2) and (4) if and only if the $t$ variable separates the cluster’s points.

3.3. Practical choice of $g(t)$

After Section 3.2, the only theoretical requirements for the $g$ function are that the function is continuous and strictly monotonic. Now, depending on this function, successive powers which are used in the matrix $S$ can rapidly go to very high or very low values. Hence, one must choose the $g$ function in such a way that none of the coefficients of $S$ tends to infinity as the power (i.e. the row number) increases. This means that $-1 \leq g(t) \leq 1$, for all possible values of $t$. A solution is $g(t) = 2F(t) - 1$, where $F(t)$ is a continuous strictly monotonic approximation of the cumulative probability function of $t$. Another advantage of this solution is that the distribution of $g(t)$ is approximately uniform in $[-1, 1]$. Note that one can as well choose $g(t) = F(t)$, and then the variation interval is $[0, 1]$.

3.4. Summary of the method

For the whole application:

- choose one of the variables, which is the most probably separating, as the variable $t$,
- determine a function $g(t) = 2F(t) - 1$, or $g(t) = F(t)$, where $F(t)$ is a strictly monotonic continuous approximation of the cumulative probability function of $t$.

Encoding a cluster:

- form the matrix $S = (s_{ij}), s_{ij} = [g(t_j)]^{i-1}$,
- the first column of the code is $C_1 = T$, in increasing order of $t$ values,
- the $i$th column of the code is $C_i = SV_i, 2 \leq i \leq n$.

Decoding a code matrix:

- $T = C_1$,
• if all components of \( T \) have distinct values then the code is decodable, and one forms the matrix \( S \) as for the encoding,
• \( V_i = S^{-1}C_i, 2 \leq i \leq n. \)

Note: If certain values of \( t \) are very close, then the matrix \( S \) is ill-conditioned, which implies that a small approximation error of \( C_i \) can result in a quite large approximation error in the decoded \( V_i \).

3.5. Computational test

Given that the above described encoding satisfies requirements (1) and (3) in all cases, various topological properties of data spaces are preserved in code spaces. However, since this encoding does not satisfy requirement (2) everywhere, there is a potential drawback which is that two distinct clusters can have the same code, and combined with (3), there are quite different clusters which have close codes. Now, since the set of unseparated clusters is of zero measure, one can hope that the drawback is limited and has statistically weak effect. For the computational test, we generated 20 uniformly random clusters of points in \([-1, 1]^m\) (i.e. \( n = 16 \)). Ten modified versions of each cluster were generated by adding a random quantity, uniformly sampled in \([-\delta, \delta]\), to each coordinate of each point. The modification scale \( \delta \) was experimentally varied from 0.0001 to 1.28, and each added quantity was eventually resampled until the modified coordinate was in \([-1, 1]\). The distance between the code \( C \) of a cluster and the code \( C' \) of a modified version was defined as the Euclidean matricial norm of the difference of codes: \( d(C, C') = \|C - C'\| \), which is equivalent to the Euclidean distance in \( R^m \). We also computed, for comparison, the Hausdorff distance between clusters in the data space. Mean distances (and standard deviations) are reported in Table 2.

As one can see in Table 2, the distance of codes is, on the average, a monotone increasing function of the modification scale, as is the Hausdorff distance (however the relations are not linear). Hence, it seems that the encoding method has reasonable properties for practical use. Now, it is clear that when one chooses learning examples for a neural algorithm, it is desirable to avoid those examples of clusters whose first coordinate is not separating. When decoding an approximated output code, one has to consider with caution clusters whose points are very irregularly spaced on the first coordinate (in fact, what is important is the spacing of \( g(t) \) values).

3.6. An example of pseudo-sequence problem

One can suspect that many natural processes are in fact pseudo-sequences, that is sequences of randomly ordered events or random mixtures of several distinct processes. In order to show how the above cluster encoding method can help to solve pseudo-sequence problems, we take now the example of sequences made of a random mixture of two independent processes: a process generated by the well-known logistic map, and a process generated by the so called ‘kappa map’ (Husmeier & Taylor, 1998). A logistic process will be defined by:

\[
y(t + 1) = \alpha y(t)[1 - y(t)], y(0) = 0.5, \alpha \in [3, 4],
\]

while a kappa process will be defined by:

\[
z(t + 1) = 1 - z(t)^2, z(0) = 0.5, \kappa \in [0.5, 1.25].
\]

Taking \( t = t' + t'' \), one can define a pseudo-sequence by:

\[
x(t + 1) = y(t' + 1) \text{ with probability } p(t', t'', \Delta_{\text{max}}),
\]

\[
x(t + 1) = z(t'' + 1) \text{ with probability } 1 - p(t', t'', \Delta_{\text{max}}),
\]

where the probability of processes is given by:

\[
p(t', t'', \Delta_{\text{max}}) = \max(0, \text{min}(1, 0.5 + 0.5(t'' - t')/\Delta_{\text{max}})),
\]

where \( \Delta_{\text{max}} > 0 \) is an integer number.

Note that if \( \Delta_{\text{max}} \) tends to infinity then \( p = 0.5 \) at any step, independently of previous events, while taking a low value for \( \Delta_{\text{max}} \) constrains \( t' \) and \( t'' \) to remain close to each other. This is a way of limiting the random variability of the mixture. However, the global probability of each of the two processes is always 0.5.

A sequence can be considered as a cluster \{\((t, x(t)); 1 \leq t \leq m\)\} in \( R^2 \), where \( t \) is obviously a separating variable uniformly distributed in the interval \([1, m]\). Then one can take \( g(t) = th/m \) and apply the cluster encoding method. In this study, we take \( m = 200 \), and we can draw the sequences given that they are in \( R^2 \). Note, however, that the same methodology can as well be applied with higher dimension (vector sequences), but this would be hard to draw.

Each of the two component processes of a sequence depends on a parameter, and each pair of parameters \((\alpha, \kappa)\) corresponds to a family of random mixtures of the same processes, while changing the parameters leads to another family (see Fig. 2). The question is: does the cluster code reflect the family the sequence belongs to? If this is the case, then the distance between codes of sequences belonging to the same family must be lower than the distance between codes of sequences belonging to distinct families,
and the encoding method can be used for building robust sequence classifiers (for example). Of course, cluster codes are useful only if they reflect the component processes better than data themselves.

A code distance can be defined, as in Section 3.5, by the Euclidean norm of the difference of codes, while a data distance can be defined in the same way, replacing the code components by the (sequentially ordered) data points. Note that, in the present case, distances do not depend on the values of \( t \), since these values are the same for all sequences.

A computational experiment was designed as follows. Four values of \( D_{\text{max}} \) were selected (2, 3, 4, and 200). For each of these values, a uniform random sample of 80 pairs of parameters \((\alpha, \kappa)\) was generated in the sampling intervals of these parameters. 40 of these pairs were used for generating 40 pairs of sequences \((m = 200)\), the two sequences in a pair being distinct random mixtures of the same two processes. The remaining 40 pairs of parameters were randomly associated to the previous ones for generating 40 other pairs of sequences, the two sequences in a pair being random mixtures of (randomly) distinct processes. The data distance and the code distance were computed for each pair of sequences. Mean distances (and standard deviations on 39 degrees of freedom) for all experimental conditions are reported in Table 3. In addition, Student \( t \) tests (with 78 degrees of freedom) were used for comparing the distances between sequences generated with similar processes to the distances between sequences generated with distinct processes. Statistical significance was tested according to the usual decision threshold \( p < 0.05 \) (while ‘n.s.’ means that the tested difference is statistically non significant).

As one can see in Table 3, with \( D_{\text{max}} = 2 \), the data distance was just significantly lower for similar processes than for distinct processes, while for higher values of \( D_{\text{max}} \), the data distance clearly does not allow for detecting the similarity of processes. Contrasting with this result, the code distance was always largely and significantly lower for similar processes than for distinct processes. Hence, the proposed encoding method clearly appears as an efficient tool for solving pseudo-sequence problems. Moreover, this
result suggests that the tool allows for robust comparisons of sequences in general.

4. Conclusion

Two methods for encoding clusters were presented. In applications, cluster codes can be used for comparisons or as arguments of various functions (Spline functions, Radial Basis functions, etc.), and decodable codes can also be used as output of approximation processes. The first method is the best one from a theoretical point of view since it satisfies all requirements specified in the Introduction, for all clusters. This method has limited applications in practice since, due to its relative complexity, it is reserved for small clusters in low dimension spaces, and applications which do not require decoding. However, it is well adapted for encoding multicluster configurations, and an example of application in sport’s science area was provided. The second method has some theoretical drawbacks, but it is very simple and concise, which makes it usable in most practical cases. There is no special limitation concerning the size or the dimension of clusters and practical examples were provided. A reviewer asked whether the presented coding schemes are local or global. The answer is that the two coding schemes have the unusual property of being simultaneously local and global. They are global in that sense that any code component depends on all points of the cluster. On the other hand, they are local in that sense that any point of the cluster can be exactly deduced from the code. Finally, one can note that there are probably other possible approaches to the problem of encoding clusters, in particular in the field of algebraic geometry, and this is a matter for further research.

Table 3
Mean data distance and code distance (with standard deviation) between sequences made of distinct random mixtures of independent logistic and kappa processes. The component processes of two compared sequences can be similar or randomly distinct. When processes are similar, lowering $\Delta_{\text{max}}$ results in bringing the ranks of similar values of the two sequences nearer.

<table>
<thead>
<tr>
<th>$\Delta_{\text{max}}$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data distance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similar ($\alpha, \kappa$)</td>
<td>3.27 (0.90)</td>
<td>4.15 (1.15)</td>
<td>3.79 (1.10)</td>
<td>3.55 (0.99)</td>
</tr>
<tr>
<td>Distinct ($\alpha, \kappa$)</td>
<td>3.66 (0.75)</td>
<td>4.10 (0.82)</td>
<td>4.07 (0.88)</td>
<td>3.80 (0.96)</td>
</tr>
<tr>
<td>Student $t(78)$, significance</td>
<td>2.10, $p &lt; 0.05$</td>
<td>0.21, n.s.</td>
<td>1.23, n.s.</td>
<td>1.12, n.s.</td>
</tr>
<tr>
<td>Code distance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similar ($\alpha, \kappa$)</td>
<td>2.66 (2.63)</td>
<td>4.06 (3.60)</td>
<td>3.15 (2.98)</td>
<td>3.76 (1.73)</td>
</tr>
<tr>
<td>Distinct ($\alpha, \kappa$)</td>
<td>9.24 (5.33)</td>
<td>9.57 (6.32)</td>
<td>10.07 (6.03)</td>
<td>8.89 (4.52)</td>
</tr>
<tr>
<td>Student $t(78)$, significance</td>
<td>7.00, $p &lt; 0.001$</td>
<td>4.79, $p &lt; 0.001$</td>
<td>6.52, $p &lt; 0.001$</td>
<td>6.70, $p &lt; 0.001$</td>
</tr>
</tbody>
</table>

Acknowledgements

This work was partially supported by a grant from Ministère de l’Education Nationale, de la Recherche et de la Technologie—ACI ‘Cognitique’ (1999, #90).

References


Barnsley, M. F. (Academic PreP). Fractals Everywhere, (2nd ed.).


