13 Self organization of color vectors

Ordered 2D projection of random 3D color vectors

Objective:

In this second example we demonstrate how we can map (project) 3D color vectors onto a 2D plane in an orderly fashion by the SOM algorithm. The color vectors are mixtures of red, green and blue colors in which the color components vary. These colors become topographically ordered on a 2D plane with respect to both intensity and hue to produce the so-called chromaticity diagram that occurs in human color vision.

13.1 Different nature of items and their attributes

Although we shall be dealing with physical attributes in this example, too, nonetheless the nature of the problem is completely different from the first case. First of all we do not have a finite set of concrete objects like the metals in the first example; *each input item is only a color shade*, and there exists an indefinite number of them in practice. On the other hand, each shade is represented only by three attributes, namely, the intensities of the three components of the basic colors, which are red, green, and blue. Nonetheless, although the mathematical representation of the input items is different from that in the previous example, the self-organization of the colors occurs formally in a similar way.

13.2 Color vectors

A three-dimensional *RGB (red-green-blue) color vector* is a *digital code* for any mixtures of shades and intensities of visible colors. In digital representation, the intensities of the three *basic color components* (R, G, B) are given as real scalars in the range $[0, 1]$.

Our aim is to show that if the input data represent quite random color vectors, the SOM algorithm is able to produce a representation of colors such that their *distribution* remains the same, but the colors become *spatially ordered* on the SOM array such that both the *hue* and the *intensity* in neighboring models change gradually; it is said that a *topographic order* of the colors has ensued. Because we can understand the relations between colors intuitively, we use this color example as an abstract model for more general *topographic self organization*.

With a proper scaling of the input vectors, an SOM can be produced that represents the generally known *chromaticity diagram* or *CIE diagram*, where the hue and the saturation of color become represented in polar coordinates like in the human vision.

Consider a MATLAB variable C(a,b,c), which represents a *three-dimensional table*. The parameters `a` and `b` are the indices of a row and a column in the table, and `c` is a vector that has three elements: its first element represents the intensity of *pure red*, the second element represents the intensity of *pure green*, and the third component is the intensity of *pure blue*, respectively.

Let us exemplify the digital coloring by the following concrete example. The MATLAB function `image(C)` defines the coloring of the square `(a,b)` by that pure color component, the intensity of which is defined by the vector `c`. If one needs *mixed colors*, the same location `(a,b)` must be painted *separately* by the intensities of the elements of `c`; in other words, the mixed colors are defined digitally by *superimposing* basic color components.

However, unlike in wet painting, where the mixture of blue and yellow produces the green color, in digital color definition the mixture of *red and green* produces the *yellow* color. Therefore yellow in the area `(a,b)` is produced by the combination of the functions `C(a,b,1)` and `C(a,b,2)`. Consider the following MATLAB commands:

```
C = zeros(1,4,3);       % 1 by 4 table, three color components

C(1,1,1) = 1;           % leftmost (1,1) area is pure red C(:,:,1)
C(1,2,2) = 1;           % second (1,2) area is pure green C(:,:,2)
C(1,3,3) = 1;           % third (1,3) area is pure blue C(:,:,3)

C(1,4,1) = 1; C(1,4,2) = 1;  % fourth area is painted pure yellow:
                             % combination of pure red = C(:,:,1)
                             % and pure green = C(:,:,2)

image(C)                     % painting
```

These commands produce the color picture in Fig. 16.

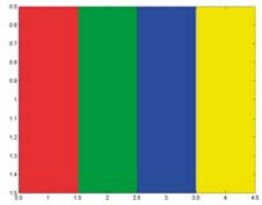

Fig. 16. Coloring example.

Another picture, in which a 25 by 25 array is colored by *random colors*, is shown in Fig. 17.

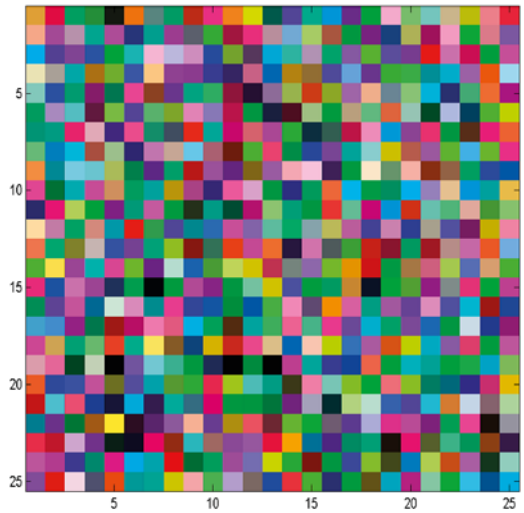

Fig. 17. Coloring of a 25 by 25 array by random color vectors.

13.3 The SOM script for the self organization of colors

In this subsection we initialize the SOM by the function `som_lininit`, *and continue with the training function* `som_batchtrain`, *which constructs the SOM of color vectors.*

As the *input data* we use 10'000 *random color vectors*, which constitute the 10000 by 3 *input data matrix* `X`. The SOM algorithm computes the data matrix `M`, which is a 625 by 3 matrix. It is to be denoted that in the SOM algorithm, for mathematical reasons, all of the model vectors are *concatenated* into a *vertical array* that has as many rows as there are nodes in the SOM array. It is not until we *display* the SOM array that we reshape the vertical array as a rectangular array, in this case 25 by 25.

Training parameters. The definition of the training parameters of the SOM is made first. We may decide to use a square SOM array (lattice) of the size 25 by 25 nodes, which is defined by the vector `msize =[25 25]`. In this simple example we define the SOM array as *rectangular*, `lattice = 'rect'`. We may prefer to use the *Gaussian neighborhood function*. Its definition `'gaussian'` follows the `'neigh'` parameter name in the function `som_batchtrain` defined below. Other parameters are the average *radius* of the neighborhood function, which decreases linearly with the coarse training cycles. In this self-organization example we have found it proper to use the initial value of 10 and the final value of 7, and during the fine training cycles let the radius decrease from 7 to 5, respectively. In other words, we did not make use of any stopping rule. The number of training cycles in both coarse and fine training shall tentatively be 50.

```
msize = [25 25];
lattice = 'rect';
radius_coarse = [10 7];
radius_fine = [7 5];
trainlen_coarse = 50;
trainlen_fine = 50;
```

Training functions. Next we define the SOM Toolbox functions `som_lininit` and `som_batchtrain`. The former carries out the *linear initialization* of the SOM. The `som_batchtrain` function is then applied twice: first for coarse training, and then for fine training. The last parameters `'shape','sheet'` in `som_lininit` mean that the topology of the SOM array is a plane.

For the training inputs `X` we use *random values* of the color vectors: `X = rand(10000,3)`.

The following commands, which form the *MATLAB SOM Toolbox script*, may now be self-explanatory. These instructions form the complete script or program which computes the SOM array of *self-organized colors*.

```
X = rand(10000,3); % random input (training) vectors to the SOM
smI = som_lininit(X,'msize',msize,'lattice',lattice,'shape', ...
'sheet');
smC = som_batchtrain(smI,X,'radius',radius_coarse,'trainlen', ...
trainlen_coarse, 'neigh','gaussian');
sm = som_batchtrain(smC,X,'radius',radius_fine,'trainlen', ...
trainlen_fine, 'neigh','gaussian');
```

Display of the SOM. This time we do not need the SOM graphic function, since we are not calibrating the SOM array in the usual way. The matrix values `M` represent color shades, which can displayed directly by the `image` function. The result is shown in Fig. 18.

```
M = sm.codebook;
C = zeros(25,25,3);
for i = 1:25
    for j = 1:25
        p = 25*(i-1) + j;
        C(i,j,1) = M(p,1); C(i,j,2) = M(p,2); C(i,j,3) = M(p,3);
    end
end
image(C)
```

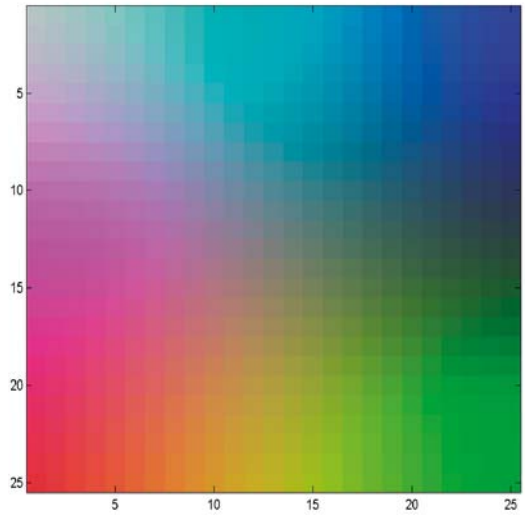

Fig. 18. Self organization of random colors in the SOM.

Computation of the chromaticity diagram. Another experiment, in which the *square root* of the previous color vectors was used as input data matrix `X`, is shown in Fig. 19.

Fig. 19 resembles the *chromaticity diagram*, which is a representation of *intensity* and *hue* of colors in two-dimensional polar coordinates. This diagram, also called the CIE diagram, is experimentally verified to exist in the human brain, in the visual cortex.

Like in the CIE diagram of colors, the pale area in Fig. 19 is also in the middle, and we have an ordered representation of intensity and hue of and mixed colors. There is no black area in this illustration like in Fig. 18. Obviously the square root emphasizes light colors to produce this result.

The square root resembles the *logarithm*, and it is known that in the biological sensory systems the subjectively experienced signal intensities are often logarithmically related. This might explain why Fig. 19 resembles the experimentally constructed CIE diagram. However, we cannot use logarithmic scales in digital signal processing, because small signals are transformed into very large negative logarithmic values.

Discussion. This example showed how a *three-dimensional color solid* was projected onto a *two-dimensional plane*. The plot includes areas where also the intensities of colors vary: for example, in Fig. 8 there is a pale area in the upper left corner, as well as a dark area on the right side. In the former, all of the numerical values of color components are close to unity, while in the latter, the numerical values of all of the components of the color vectors are low.

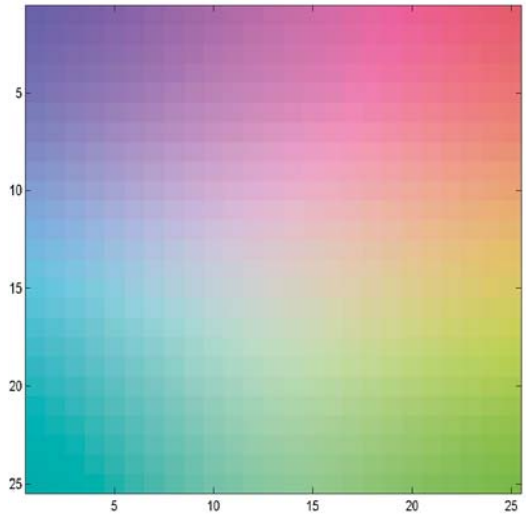

Fig. 19. Another self organization of random colors by the SOM, when the *square root* of the color vectors was used as input data to the SOM. This image resembles the *chromaticity diagram* or the *CIE color diagram of the human color vision.*

The somewhat more realistic color diagram in Fig. 19 was constructed using color vectors with an enhanced distribution of whiter shades.

Certainly this projection is *nonlinear*. One may imagine that the nodes of the SOM array form a *flexible two-dimensional network* that is trying to approximate the three-dimensional color solid. Like the *fractal surfaces*, also called *space-filling surfaces*, this "flexible" SOM network behaves somewhat in the same way. It is trying to approximate the higher-dimensional structures, but is succeeding only partly. The SOM network is more or less "stiff," depending on the *width of the neighborhood function*, and its ability of approximating higher-dimensional structures depends on the choice of this neighborhood function. In this example the final width of the neighborhood function was equal to 5, which is a rather large value compared with what is normally used. On the other hand, with this value the color maps shown in Figs.18 and Fig. 19 have become *globally ordered*, i. e., they do not contain partially disorganized areas.

One may also understand that while the SOM network is a *projection surface* onto which the higher-dimensional data distribution is projected nonlinearly, this projection surface, in the two-dimensional display, is straightened up.

This example also shows that although the SOM projection is able to illustrate higher-dimensional distributions, it is not always unique. Especially depending on the choice of the neighborhood function, but also on the scaling of the input data and different random-number sequences, the detailed projections may vary from one mapping to another. The *topographic relations* between the items, however, tend to be preserved.