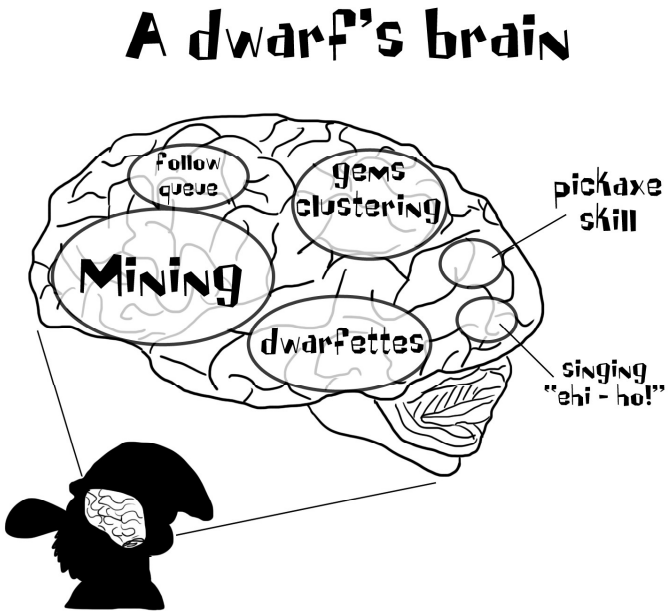
Chapter 18

Self-organizing maps

The grandmother cell *is a hypothetical neuron*
that represents any complex and specific concept or object.
It activates when a person's brain "sees, hears, or otherwise sensibly discriminates"
such a specific entity as his or her grandmother.
(Jerry Lettvin, 1969)



From the previous chapters, you are now familiar with the basic clustering techniques. Clustering identifies group of similar data, in some cases with a hierarchical structure (groups, then groups containing groups, ...). If an internal representation is available, a group can be represented with a prototype. This chapter deals with **prototypes arranged according to a regular grid-like structure and influencing each other if they are neighbors in this grid**.

The idea is to **cluster data (entities) while at the same time visualizing this clustered structure on a two-dimensional map**. One wants a visualization that is at least approximately coherent with the clustering — this should be puzzling enough to continue reading.

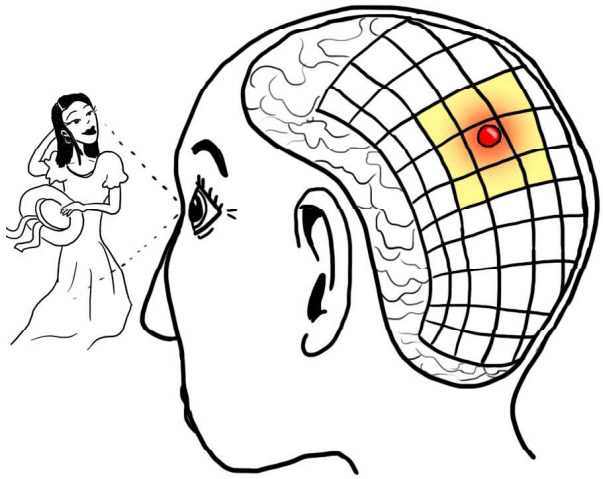


Figure 18.1: The presence of certain external stimuli activates a region in the brain ("*grandmother cell*"). In some areas, neurons are approximately organized according to a two-dimensional structure, like in the cortex, the surface layer of the brain where most high-level functionalities are located.

Let each cluster i be associated with a representative vector, a prototype $\mathbf{p}_i$. In the field of marketing, it is usual to identify different customer types, and describe them through prototypes (wealthy single individual, middle-class worker with family, etc.). A prototype will have the same number of coordinates as our entities and each component of the vector will describe a representative value for the given cluster, for example the average value of the entities contained in the cluster.

A coherent visualization demands that similar prototypes are placed in nearby positions in the 2D visualization space. Of course, for high-dimensional problems (problems with more than two coordinates), no exact solution is available and one aims at approximations which are sufficient for us to reason about the data. A **self-organizing map (SOM)** is a type of artificial neural network that is trained by using unsupervised learning to produce a two-dimensional representation of the training samples, called a map. This model was introduced as an artificial neural network by Teuvo Kohonen, and is also called a **Kohonen map**.

18.1 An artificial cortex to map entities to prototypes

A self-organizing map consists of component nodes or neurons. The arrangement of nodes is a regular placement in a two-dimensional grid. In some cases the grid is hexagonal, so that each node has six closest neighbors instead of four neighbors like in a traditional squared grid (Fig. 18.3). Associated with each node i is a prototype vector $\mathbf{p}_i$ of the same dimension as the input data vectors, *and* a position in the map space.

The analogy is again with our neural system: the **neurons are organized according to a physical network of connections in the brain, in practice two or three-dimensional**. Some neurons are tuned by evolution and training to fire electrical signals for particular events, as shown in Fig. 18.1. For example, a neuron may fire when your mother enters your visual field. The prototype is in this case given by visual features corresponding to your mother, the position is the physical position of the neuron in the brain.

Another principle of our neural systems is that, in many cases, **neurons that are neighbors tend to fire for similar input data** (a neighbor of the neuron firing for your mother presence may be close to one firing

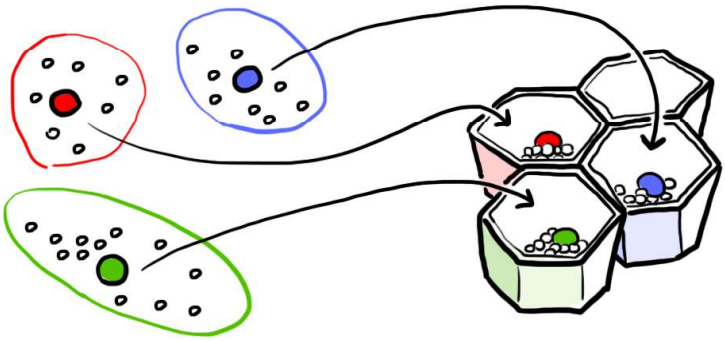


Figure 18.2: A SOM maps entities in a multi-dimensional space into cells in two-dimensional space. Each cell contains a prototype and the entities for which the prototypes is the most similar one.

for an old photo of your mother). After training, the self-organizing map describes a mapping from a higher-dimensional input space to a two-dimensional map space. Each cell in two dimensions corresponds to a neuron and contains a prototype vector. A generic entity is then mapped (or assigned) to the neuron with the prototype vector which is *nearest* to the vector describing the entity, as shown in Fig. 18.2. The training can start from a random initial configuration of the prototype vectors (for example picked to be equal to a random subset of entities) and then iterate by presenting and mapping randomly selected cases. The winning neuron $c(\mathbf{x})$, or c for brevity, is identified as the one with the prototype closest to the vector describing the current case $\mathbf{x}$:

$$c(\mathbf{x}) = \arg\min_i \|\mathbf{x} - \mathbf{p}_i\|. \tag{18.1}$$

Then the winning prototype $\mathbf{p}_{c(\mathbf{x})}$ is changed to make it more similar to the one of the current case presented to the network. In addition, the prototypes of *nearby* vectors are also changed in a similar manner, although by smaller and smaller amounts as the distance in the grid increases.

Think about a democratic system in which voters (entities) are asked to educate a set of regularly-arranged representatives (like in a chamber of the parliament) so that at least one of them represents a cluster of related ideas, and in which representatives sitting in nearby positions influence each other, and tend to become similar. Two "force fields" are active, attractive forces between entities and prototypes, and attractive forces between neighboring prototypes on the grid. Entities (voters) compete for prototypes: each entity is pulling its *winning* prototype and, to a less extent, the neighbors of the winning prototype, to move a bit towards itself, to make it progressively more similar. Of course, different entities are pulling in different directions and the resulting dynamical system is very complex.

After explaining the basic mechanism and motivations, let's now fix the details. In an online learning scheme, at each iteration t a random entity **x** is extracted, its winning neuron c is determined, and all prototype vectors $\mathbf{p}_i(t)$ at iteration (time) t are then modified as follows:

$$\mathbf{p}_i(t+1) \leftarrow \mathbf{p}_i(t) \; + \; \eta(t) \cdot \mathrm{Act}\big(c(\mathbf{x}), i, \sigma(t)\big) \cdot (\mathbf{x} - \mathbf{p}_i(t)), \tag{18.2}$$

where $\eta(t)$ is a time-dependent small learning rate, $\mathrm{Act}\big(c, i, \sigma(t)\big)$ is an **activation function** which depends on the distance between the two neurons in the 2D grid, and on a time-dependent radius $\sigma(t)$. The two neurons involved in the formula are: the winning neuron c for pattern $\mathbf{x}$, and the neuron i whose pattern $\mathbf{p}_i(t)$ is being updated. The modification mechanism resembles the update described in equation (16.9) for soft clustering in k-means, but with the important difference given by the regular two-dimensional organization of the neurons, which now determines the activation level.

To help convergence, usually the learning rate decreases in time, and the same happens for the radius parameter. The idea is that at the beginning the neuron prototypes are moving faster (*neural plasticity* is higher for young children!), and they tend to activate a larger set of neighbors, while movements are smaller and limited to a smaller set of neighbors in the last phase, when hopefully the arrangement already identified the main characteristics of the data distribution and only a fine-tuning is needed. The learning rate in some cases decreases like $\eta(t) = A/(B + t)$. A reasonable default can be $\eta(t) = 1/(20 + t)$.

In batch training, all N entities $\mathbf{x}_j$ are presented to the SOM and their winning neurons $c(\mathbf{x}_j)$ are identified before proceeding to an update as follows:

$$\mathbf{p}_i(t+1) \leftarrow \frac{\sum_{j=1}^{N} \mathsf{Act}\big(c(\mathbf{x}_j), i, \sigma(t)\big) \cdot \mathbf{x}_j}{\sum_{j=1}^{N} \mathsf{Act}\big(c(\mathbf{x}_j), i, \sigma(t)\big)}. \tag{18.3}$$

Each prototype is updated with a weighted average over all entities, the weight being proportional to the vicinity in neuron grid space (usually two-dimensional) between the winning neuron prototype and the current prototype.

Because of the system complexity, one is encouraged to try different parameters and different time schedules until acceptable results are obtained. For example, a suitable neighborhood activation function can be:

$$\mathsf{Act}\big(c, i, \sigma(t)\big) = \exp\left(-\frac{{d_{ci}}^2}{2\,\sigma^2(t)}\right), \tag{18.4}$$

where d_{ci} is the distance between the two neurons in the two-dimensional grid, and $\sigma(t)$ is a neighborhood radius, at the beginning including more neighbors than the closest ones, at the end including only a set of close neighbors. Be careful not to confuse the distance between neurons in the grid, as shown in Fig. 18.3, with the distance between prototype vectors in the original multi-dimensional space of the data!

Let $\mathsf{TOT}_{\mathsf{SOM}}$ be the total number of SOM neurons, and $\mathsf{TOT}_{\mathsf{iter}}$ the number of iterations executed. The default value starts from $\sqrt{\mathsf{TOT}_{\mathsf{SOM}}}$, a value close to the radius of the grid if the grid is a square one, and ends with the value 2, as follows:

$$\sigma(t) = \frac{(\mathsf{TOT}_{\mathsf{iter}} - t)\sqrt{\mathsf{TOT}_{\mathsf{SOM}}} + 2t}{\mathsf{TOT}_{\mathsf{iter}}}. \tag{18.5}$$

One should not be discouraged by the complexity intrinsic in this and in similar mapping tasks: in many cases acceptable results are obtained by considering simple default values for the basic parameters. On the other hand, it is not surprising that a basic mapping mechanism of our brain is indeed characterized by a high complexity level, we are intelligent and in part unpredictable human beings, aren't we?

18.2 Using an adult SOM for classification

Even if you do not want to indulge in the *debauchery of indices* of the above mathematical details, you can still use SOM effectively as a guide in reasoning about your problem. After training, the SOM can be used to classify new objects by finding the closest (winning) prototype and assigning the new object to the corresponding cell, as illustrated in Fig. 18.4. In many cases, after looking at the prototypes, it will be easy to give *names* to the different cells, to help reasoning and remembering. But let's note that cells may discover *unusual* combinations, leading to interesting insight and *detection of novel groups*, and not only to re-discovering trivial classifications.

Let's imagine that you trained your SOM on marketing data: each cell may represent a characteristic group of customers. Possible names could be: "wealthy-single-individual," "poor-family-with-kids," "senior-retired-person," "spoiled-adolescent," etc. When a new customer arrives, you may easily identify the appropriate prototype, for example to pick the best strategy to sell him your products. If you are a movie fan, and

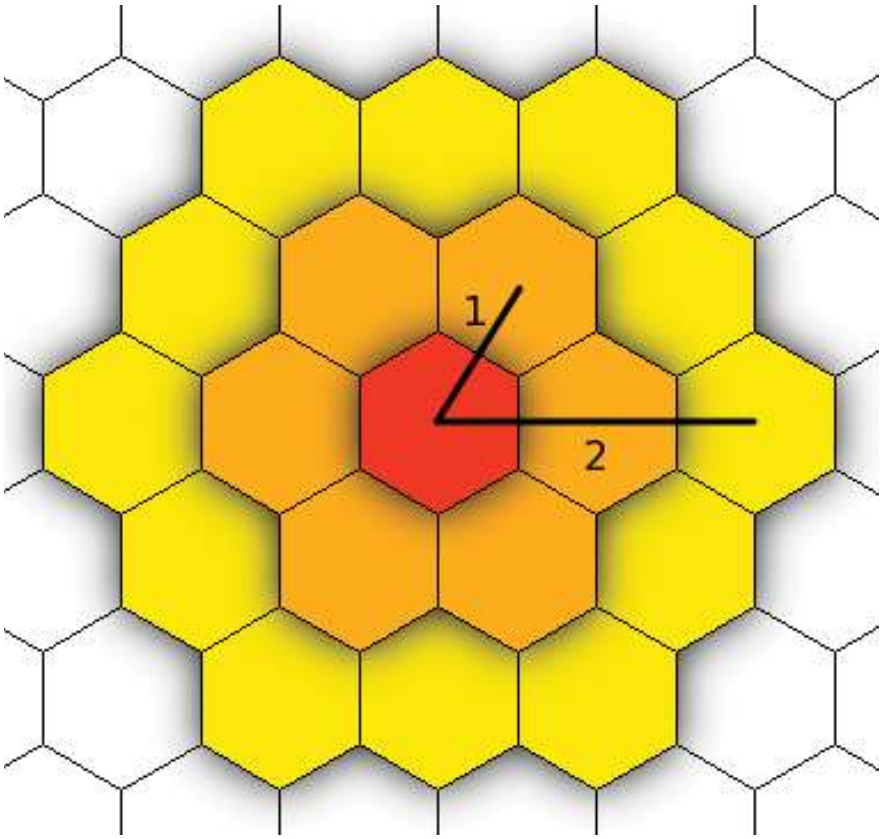


Figure 18.3: An example of neighborhood in a self organizing map: neighboring cell at distance 1 and 2 from the central are shown.

you train your SOM to classify different kinds of movies, you may use a SOM to classify a new movie, for example to predict if you will like it or not (with a high probability).

In a SOM, the quality of training can be measured by the **quantization error** (the average error incurred by substituting an entity $\mathbf{x}$ with its winning prototype vector $\mathbf{p}_{c(\mathbf{x})}$, i.e., the average distance from each data vector to its nearest prototype) or by the **topological error**, which is related to the failure to assign close vectors in the original high-dimensional space to close neurons in neuron-grid space (usually two-dimensional). The topological error can be evaluated as the fraction of all data vectors for which the first and the second nearest prototypes (in the original multi-dimensional space) are not represented by adjacent neurons in the grid mapping.

Color coding can be used to represent the value of the data point along a dimension, while the size of each hexagon can represent the value along another dimension (Fig. 18.5). The colored maps are called components or **component planes** and can be compared to identify local relationships. One can devise interesting new analysis techniques by combining the SOM map with a scatterplot display, or with a parallel coordinates plot (Fig. 17.5). For example, when the mouse pointer is moved over the SOM cells, the position of the prototype vector associated with each cell can be shown in a scatterplot or in a parallel coordinates plot. In this manner, details of relevant entities can be further analyzed.

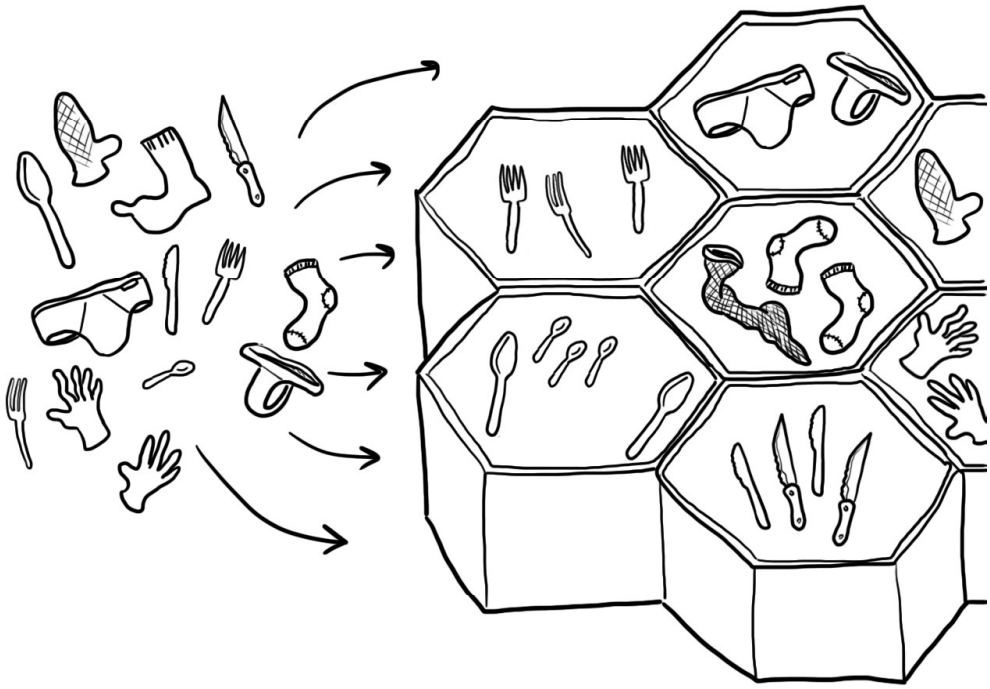


Figure 18.4: An analogy for the SOM: each cell is like a drawer in a well-organized piece of furniture. Neighboring drawers are used to contain similar objects.

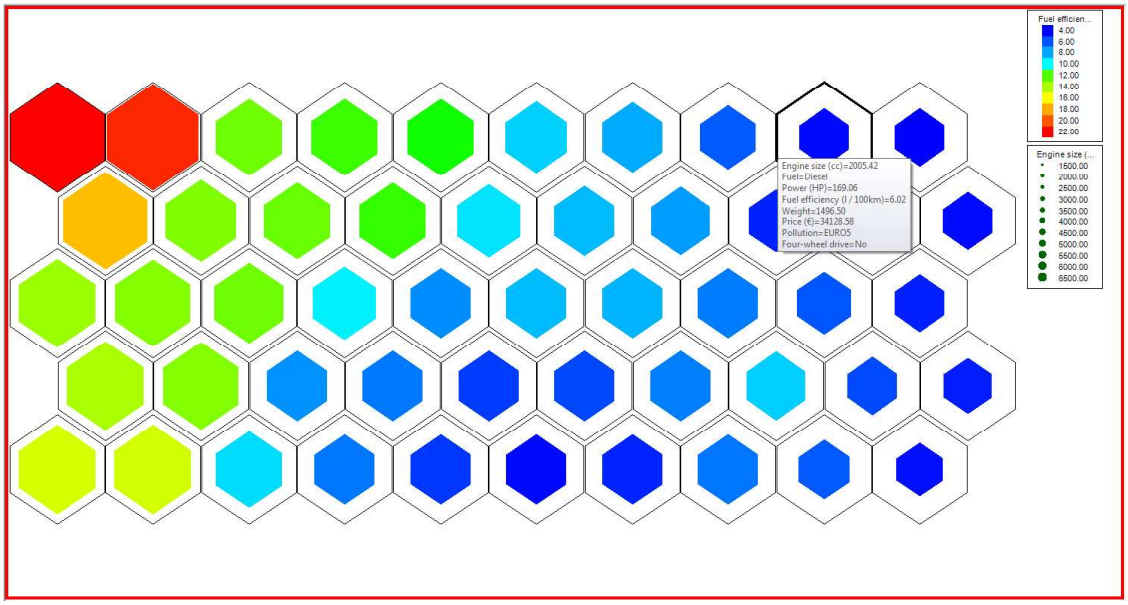


Figure 18.5: A SOM, color and size depends on two coordinates of the prototype vectors. Prototypes can be examined by passing with the mouse over the cell (LIONoso.org software).

Gist

Self-organizing maps reach two objectives: placing a set of prototypes close to clusters of data points, and having the prototypes organized in a two-dimensional grid so that neighboring prototypes in the grid tend to be mapped to similar data points.

The motivations are in part biological (our neural cortex is approximately organized according to two- and three-dimensional arrangement of neurons) and in part related to visualization. A two-dimensional grid can be visualized on the screen, and the characteristics of the prototypes are not randomly scattered but slowly varying, because of the neighboring relationships, leading to more intelligible visualizations.

If data points are imagined as schooling fishes in the sea, a SOM is an elastic fisherman's network aiming at capturing the largest number of them without breaking up.