Implementation of a simple Kohoen Network application

Federico Lusiani - matricola 567432

Neural Networks course, Academic Year 2019/2020

Date: 25/05/2021

1 Introduction

The projects aims to implement an interactive Kohoen Network, a type of SOM (Self Organizing Map). Through the GUI the user can insert data, and visualize the network as it learns. Kohoen Networks are usually used to perform dimensionality reduction and normalization on data. In this project, the network maps a 2D input on a grid topology of nodes (therefore, it performs no dimensionality reduction, just data normalization).

2 Model

A self-organizing map (SOM) or self-organizing feature map (SOFM) is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space.

In this implementation, a fixed grid topology has been chosen for the network, with a configurable number of units. In the grid topology, it is possible to plot the U-matrix of the SOM, which shows the distances of the network units between each other in input-space, giving an idea of how the grid "stretches" in the input-space.

3 Algorithm

Algorithm 1 shows the pseudo-code of the network training.
The algorithm requires two distance functions: a *inputDist* function to compute distance between points in the input space (in my implementation, an $\|\cdot\|_2$ norm) and a *topologyDist* function to compute distance between two units of the network (in my implementation, a discreet $\|\cdot\|_\infty$ norm over the grid topology).

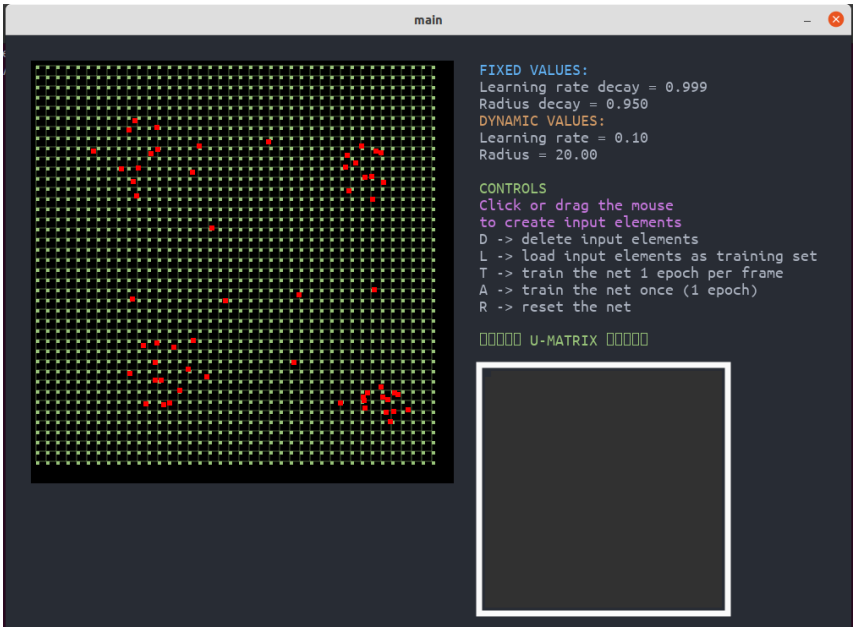
Algorithm 1 Kohoen Network - training algorithm

Given dataset array $\mathbf{D}$, weights array $\mathbf{W}$, starting radius r, radius decay factor r_decay, learning rate l and learning rate decay factor l_decay, distance functions $inputDist$ and $topologyDist$
while $stopping\_condition()$ is $False$ **do**
  for all $d \in \mathbf{D}$ **do**
    find $v \in \mathbf{W}$ that minimizes $inputDist(\mathbf{W}[v], d)$
    for all w s.t. $topologyDist(w, v) < r$ **do**
      $\mathbf{W}[w] = \mathbf{W}[w] + l * (d - \mathbf{W}[w])$
    end for
    $r = r\_decay * r$
    $l = l\_decay * l$
  end for
end while

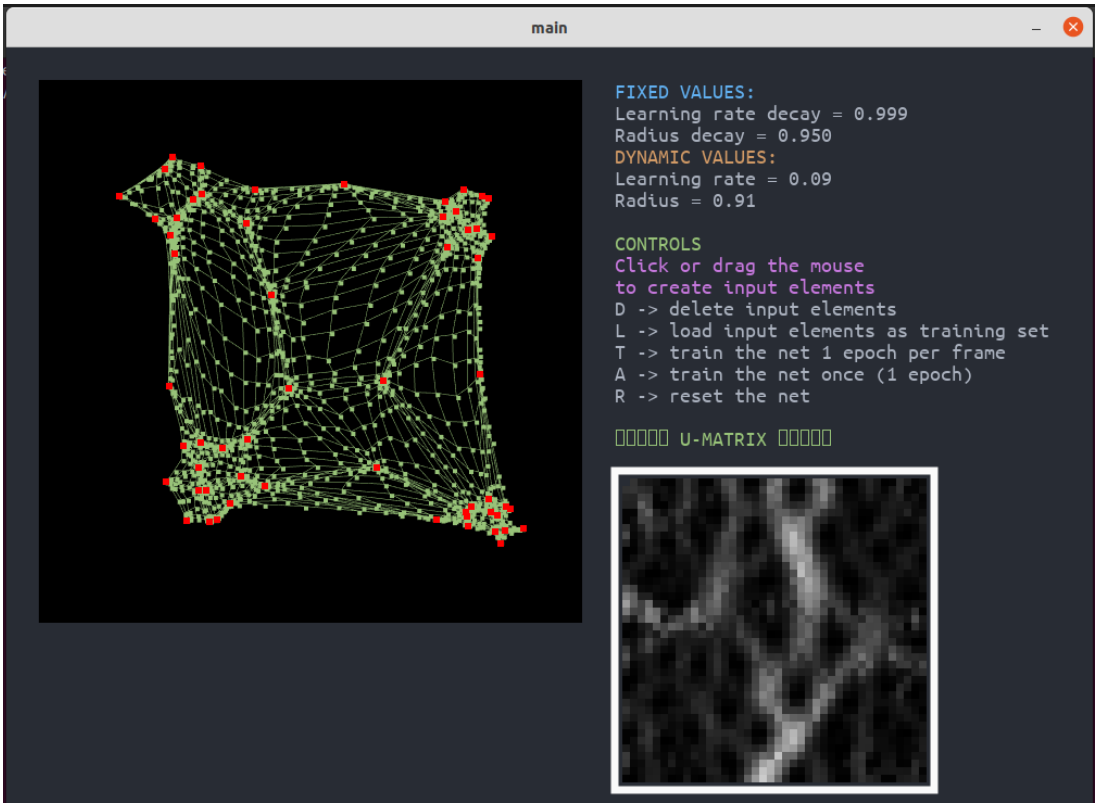
Note that the *stopping_condition* can vary depending on the application (a common choice can be setting a maximum number of epochs). Note that in my implementation it is the user that decides (via the GUI) whether the net continues the training or stops (resets).

4 GUI and results

Figure 1 shows an example of KN training. The application shows the network (green) and the user-defined input points (red) in the input-space.
The network weights are initialized as equally spaced points in a grid pattern, which can be seen as the "natural" configuration for a grid topology. As the net trains, the network units adapt to the training set, "stretching" the grid in the attempt to follow the distribution of the training set.
The GUI also shows the U-matrix of the network: a graphic representation of the euclidean distance between the network units in the input-space: light cells represent pairs of neighbouring units that are further apart, while dark cells represent pairs of neighbouring units that are closer together in the input-space. Therefore, the U-matrix offers a powerful tool to inspect the network, when the input-space has many dimensions and cannot be easily visualized directly.
Darker areas in the U-matrix indicate a cluster in the network units (and therefore in the training data as well), separated by lighter areas (representing "stretched" parts of the network between these clusters). This can be seen in Figure 1, where a cross-shaped light area separates the U-matrix in the four darker areas, corresponding to the user-drawn clusters. The same behaviour can be observed in the example of Figure 2, with 3 user-defined clusters.

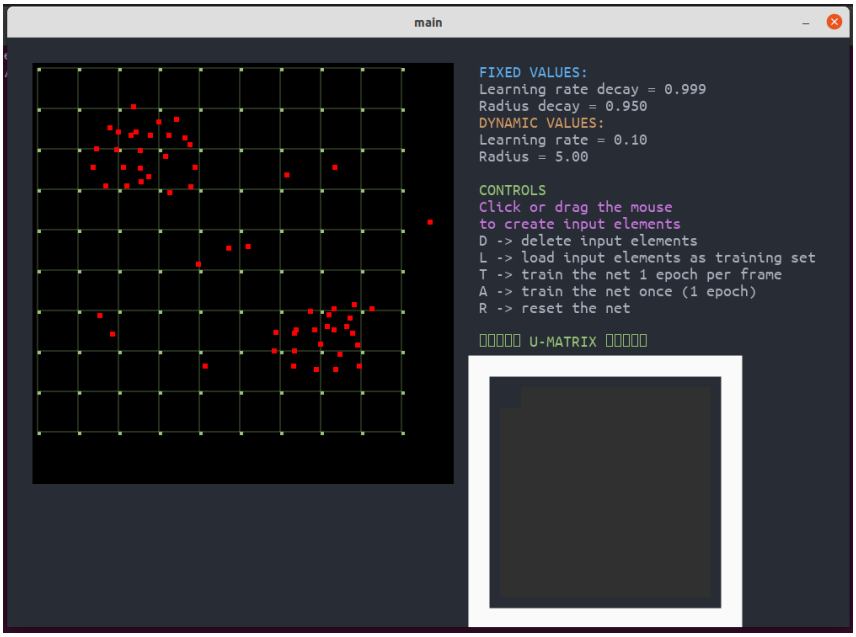


(a) Before training

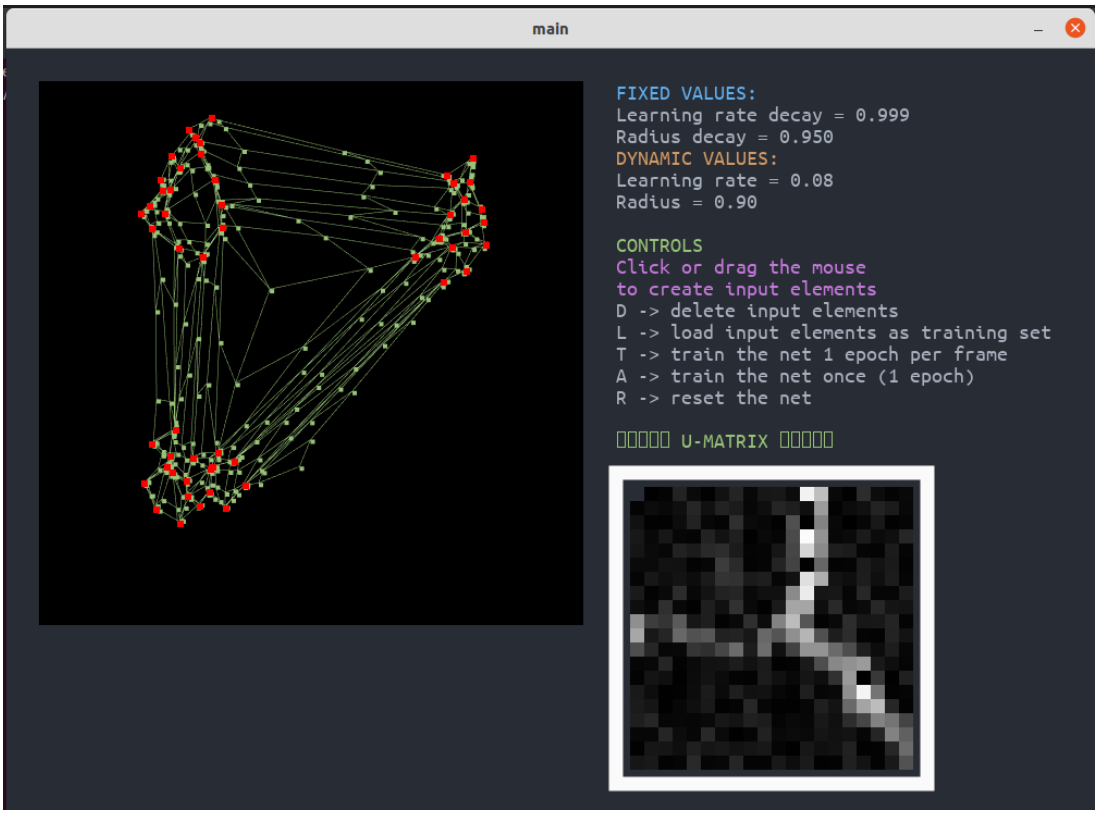


(b) After training

Figure 1: GUI frames showing the state of the network before and after the training process



(a) Before training



(b) After training

Figure 2: another example, using a smaller number of network units