Image2Map:
A Self-Organizing Map for Image Classification

Nelson Aloysio Reis de Almeida Passos

August 2025

1 Introduction

A Self-Organizing Map (SOM) [Koh82] is a type of artificial neural network for unsupervised learning, usually employed to project high-dimensional data onto a lower-dimensional space. The map is a grid of neurons (units), trained through competitive learning: for each input sample, a best matching unit (BMU), i.e., with the lowest distance to the data and therefore more similar to it, is identified. The weights of the BMU and its neighboring units are adjusted to reduce the distance between them and the sample, while a dynamic learning rate and a predefined neighborhood function control the extent of the adjustments. This process results in a mapping of the input space onto the output space.

In this project, we explore SOMs for image clustering applications. The core idea is to train a model on a dataset of two-dimensional images, where each input is represented as a vector of pixel values, and then employ it to map (predict) new images to the grid, assigning them to the nearest neuron. The goal is to showcase a simple and efficient method for image classification that does not require labeled data and can be easily visualized and interpreted. An additional interface is provided to allow the user to interact with the model before and after training and visualize the results of the clustering process.

This document is intended as a technical report of the Image2Map software, developed in Python, using the NumPy library for the algorithm implementation and the Pillow library for reading the input image files. The visualizations shown in this document were generated with Matplotlib. A Jupyter notebook to reproduce the examples is available online, and an optional graphic user interface developed with the Streamlit library is also included, providing a simple and intuitive way for users to interact with the model. No further external libraries are used, so the code can be easily run on any machine with Python or Docker installed. The source code is available for download on a GitHub repository[1].

[1] Available at: `https://github.com/nelsonaloysio/image2map`.

2 Background

Developed by Teuvo Kohonen in the 1980s, a SOM is a type of artificial neural network that uses a competitive learning algorithm to produce a low-dimensional representation of the input space, making them particularly well-suited for visualizing high-dimensional data in a lower-dimensional space, i.e., a "map".

SOMs mimic the way the human brain organizes and processes information, similarly to a Hebbian paradigm ("neurons that fire together, wire together"). More specifically, SOMs learn to adjust their weights based on the similarity of the input data, reinforcing connections between neurons that respond to similar patterns — in a similar fashion to how the neurocortex processes information in the brain. This strategy allows them to capture the underlying structure of the data and to generalize well to new, unseen examples, making them a powerful tool for unsupervised learning for diverse applications, such as dimensionality reduction, clustering and classification, and data visualization tasks.

Since its proposal, this type of neural network has been successfully applied to various fields, such as image downsampling and speech recognition, and have in this way demonstrated their effectiveness in handling complex, high-dimensional data in real-world scenarios. Other researchers extended on it by, e.g., exploring new training strategies and architectures, with variants such as the Growing Self-Organizing Map (GSOM) [RMD02], which allows the map to grow dynamically during training, and the Neural Gas algorithm [Fri94], which uses a different update rule for the weights of the neurons.

3 Algorithm

The learning algorithm of a SOM can be broken down into four main steps:

1. **Initialization:** The weights of the neurons are randomly initialized.
2. **Competition:** For each input vector, the neurons compete to become activated. The neuron with the smallest distance to the input vector is declared the best matching unit and winner of the competition.
3. **Cooperation:** The BMU and its neighbors are updated to better represent the input data. A neighborhood function is used to determine the extent of the update, with closer neurons receiving a larger adjustment.
4. **Adaptation:** The learning rate and iteration radius are decreased over time, resulting in smaller weight updates on each iteration over the input data and allowing the network to converge to a stable solution.

The algorithm stops when a predefined number of iterations is reached or the radius and learning rate converge (i.e., become sufficiently small). We describe each step in more detail next and provide examples to illustrate the concepts.

3.1 Initialization

Each unit in the SOM is assigned a weight vector of the same dimensionality as the input data. On initialization, the weight vectors are typically assigned small random values, e.g., drawn from a uniform or normal (Gaussian) distribution. This step helps to break symmetry, allowing the network to explore the input space more effectively during training and possibly leading to faster convergence.

The initial weight vectors therefore may be represented as follows:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_m \end{bmatrix}, \mathbf{w}_m = [x_1, x_2, \ldots, x_n] \quad \text{and} \quad x_i \sim \mathcal{U}(-\epsilon, \epsilon), \tag{1}$$

where ϵ is a small positive constant that determines the range of the initial weights, e.g., $\epsilon = 1$. Inputs are usually normalized to the same range, while maintaining unit variance and zero mean, to ensure all features contribute equally to the distance calculations during the learning phase.

In SOMs, in addition to a weight vector, each neuron is also assigned a fixed position in the output space, which remains fixed throughout the training process. This output (embedding, latent) space or "map" typically assumes a one, two, or three-dimensional topology, depending on the nature of the input and the specific application the model is being trained for: image processing tasks, such as converting color images into black and white, for example, may benefit from a 2D output map in spite of their (3-dimensional) RGB input.

The neuron positions and connections in the output space influence both the learning dynamics and the expected quality of the resulting map — the choice of topology impacts the input patterns learned by each neuron, and how well they generalize. Units near the border of the map may be connected to long-range neighbors, mimicking a toroidal structure, e.g., a ring or a mesh in case of one- and two-dimensional topologies, respectively, as illustrated on Figure 3.1:

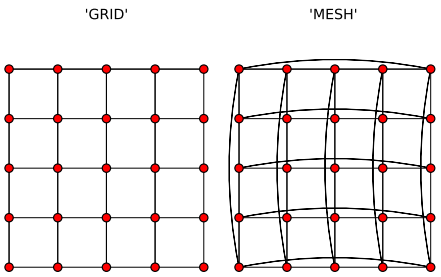

Figure 1: Example of two-dimensional topologies for the output map of a SOM. In the mesh-like structure, the farthest units in the map are also connected.

3.2 Competition

When an input vector is presented to the SOM, each neuron computes the distance between its weight vector and the input vector. The neuron with the smallest distance is declared the best matching unit (BMU) and is selected as the winning neuron, for which the weight update will be the largest.

This implementation employs cosine similarity to identify the BMU, which measures the cosine of the angle between two non-zero vectors. It is defined as the dot product of the vectors divided by the product of their magnitudes:

$$\mathrm{BMU}_i^{(t)} = \arg\max_j \left\{ \frac{\mathbf{w}_j^{(t)} \cdot \mathbf{x}_i}{||\mathbf{w}_j^{(t)}|| \cdot ||\mathbf{x}_i||} \right\}, \tag{2}$$

where $\mathbf{w}_j^{(t)}$ is the weight vector of the j-th neuron at time t and $\mathbf{x}_i$ is the i-th input vector. The arg max operation selects the neuron j with the highest cosine similarity to the input vector (i.e., the one with the smallest angle or Euclidean distance). Note that the division by the product of their magnitudes ensures the resulting scalar is invariant to the magnitudes of the input and weight vectors.

Next, the BMU weight is updated with a learning rule expressed by Formula 3:

$$\begin{aligned} \Delta\mathbf{w}_j^{(t)} &= \alpha^{(t)} \cdot (\mathbf{x}_i - \mathbf{w}_j^{(t)}) \\ \mathbf{w}_j^{(t+1)} &= \mathbf{w}_j^{(t)} + \Delta\mathbf{w}_j^{(t)}, \end{aligned} \tag{3}$$

where $\mathbf{w}_j^{(t)}$ is the weight vector of the j-th neuron at time t, $\alpha^{(t)}$ is the (dynamic) learning rate at time t, and $\mathbf{x}_i$ is the i-th input vector in the training loop.

3.3 Cooperation

This step involves updating the weights of the neighboring neurons to the BMU in the output space. This is achieved by employing the same weight update rule as in the competition step, with an additional neighborhood function Φ that determines the influence from the BMU on each neighboring unit:

$$\begin{aligned} \forall n \in \mathcal{N}_j^{(t)} : \Delta\mathbf{w}_n^{(t)} &= \alpha^{(t)} \cdot \Phi_n^{(t)} \cdot (\mathbf{x}_i - \mathbf{w}_n^{(t)}) \\ \mathbf{w}_n^{(t+1)} &= \mathbf{w}_n^{(t)} + \Delta\mathbf{w}_n^{(t)}, \end{aligned} \tag{4}$$

where $\mathcal{N}_j^{(t)}$ is the set of neighboring neurons to the BMU at time t.

The neighborhood function Φ may assume the form of a Gaussian function, or other forms similar to a "Mexican hat", where the excitatory and inhibitory influences of the BMU on its neighbors are modeled accordingly. In the software's implementation, a Gaussian Laplacian is employed by the default:

$$\Phi_n^{(t)} = \exp\left(-\frac{d^2}{2\sigma^2}\right), \tag{5}$$

where σ balances the resulting influence of the BMU on its neighbors. Figure 3.3 illustrates this and other possible examples of neighborhood functions:

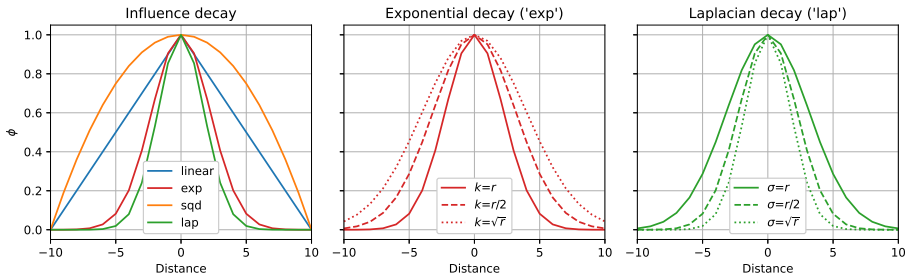

Figure 2: Example of neighborhood functions centered on the BMU (at $x = 0$). By default, $\sigma = \sqrt{r^{(t)}}$, where $r^{(t)}$ is the radius of the neighborhood at time t.

Neighboring units are selected based on their Euclidean distance by default. Figure 3.3 illustrates the effect of different distance functions on the selection of neighboring units, for a radius of $r = 2$, highlighting their possible impact.

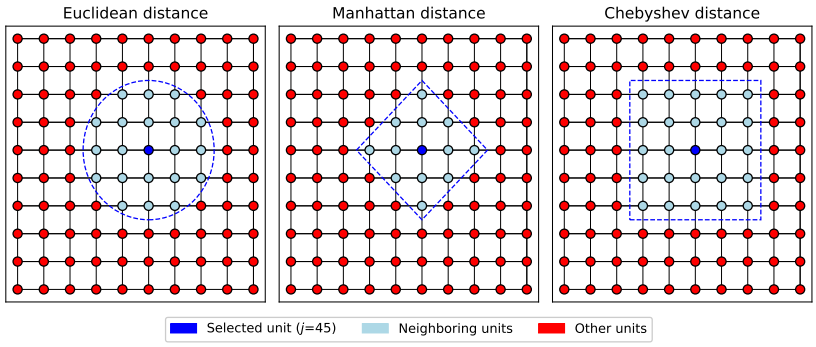

Figure 3: Impact of distance functions on the selection of neighboring units.

3.4 Adaptation

As the training progresses, both the learning rate α and the iteration radius r are typically decreased to allow finer weight adjustments with a decay function. Such decay may follow a linear, exponential, or distinct function, depending on the specific implementation and the desired convergence properties.

By default, the software's implementation employs a normalized exponential decay for both the learning rate and the iteration radius, which ensures that the updates to the weights are consistently decreasing over a preselected number of epochs. A hyperparameter allows to choose between a different decay strategy, allowing, for example, to leave the number of iterations (epochs) unconstrained. Figure 3.4 illustrates the difference between the two strategies with different decay functions (linear and exponential) with set minimum and maximum values.

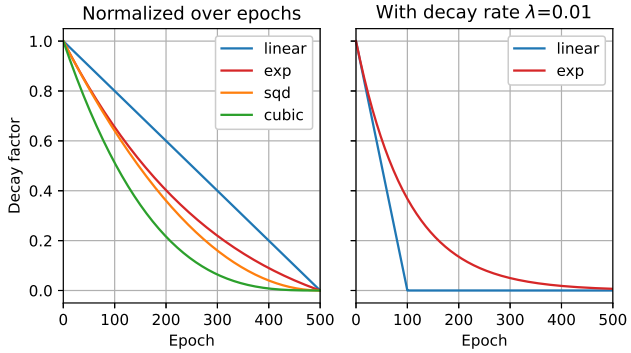

Figure 4: Decay functions with normalized (left) or fixed (right) rates over time.

4 Experiments

Two experiments were performed using the software implementation described in this document, focusing on the LINES and MNIST datasets, described next.

4.1 LINES dataset

The LINES dataset consists of a collection of 100 line drawings with 16x16 pixels, in black and white, generated for this specific software's use case. Figure 4.1 (end of the document) displays a visualization of all the inputs in the dataset.

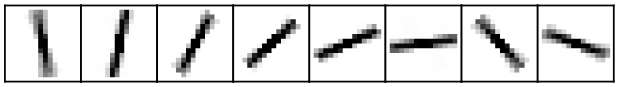

Figure 5: Output map for the LINES dataset after 100 epochs, considering a train size of 80 samples (0.8). The dataset and an animated version of this plot showing the learning process over time are available at the online repository.

We applied the SOM implementation to this dataset with a one-dimensional (line) topology consisting of $k = 8$ neurons, in order to evaluate its ability to capture the underlying structure of the data. The resulting output map after training for 100 epochs is shown on Figure 4.1, which portrays the SOM's implementation capability to organize the line drawings in a meaningful way.

4.2 MNIST dataset

The MNIST dataset consists of a collection of handwritten digits, commonly used for training various image processing systems. It contains 60,000 training images and 10,000 testing images, each of size 28x28 pixels, in black and white.

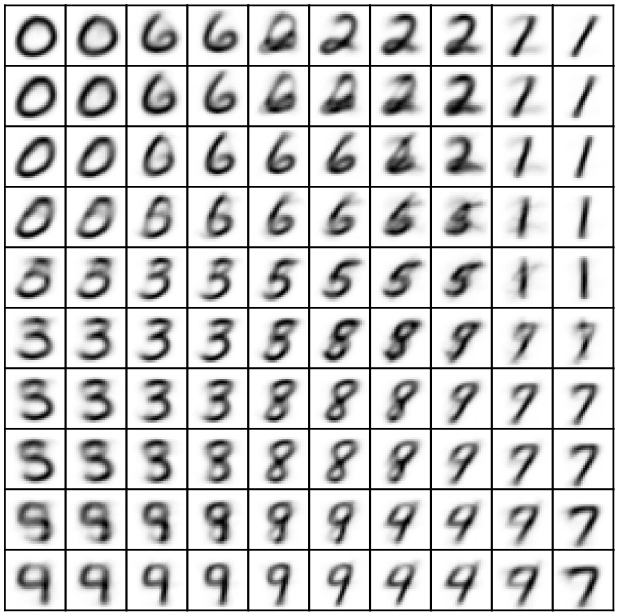

Figure 6: Output map for the MNIST dataset after 100 epochs. While the dataset is more complex than the LINES dataset, the SOM implementation still captures meaningful patterns and relationships between the different digits. Note that the original dataset colors were inverted for better visualization.

We applied the SOM implementation to this dataset with a two-dimensional (grid) topology consisting of $k = 100$ neurons, in order to evaluate its ability to capture the underlying structure of the data. The resulting output map after training for 10 epochs is shown on Figure 4.1, which illustrates the organization of the handwritten digits in the SOM. It is possible to observe distinct groups of units corresponding to different digits, demonstrating its effectiveness in capturing the complex structure of the data, in spite of its intra-class variability.

5 Conclusion

In this work, we presented an overview of the Self-Organizing Map (SOM) algorithm, including its underlying principles, mathematical formulation, and practical implementation details. We also discussed various aspects of the training process, such as competition, cooperation, and adaptation, highlighting the importance of hyperparameter tuning for optimal performance. Through the two experiments shown, we demonstrated the effectiveness of the implementation in capturing complex data distributions and facilitating their visualization. Our results underscore the usefulness of this unsupervised learning technique for a

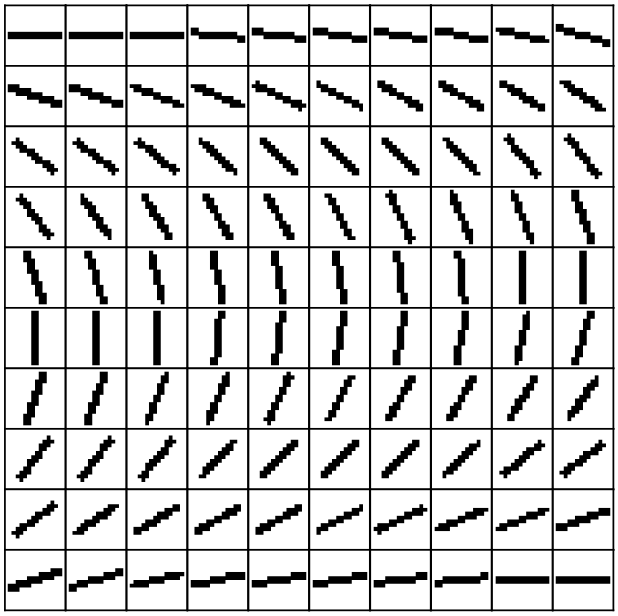

Figure 7: LINES dataset consisting of 100 line drawings in 16x16 pixels. All the images were created using the ImageMagick software (source available online).

wide range of applications, and future work may explore the integration of SOMs with other machine learning methods, as well the impact of different architectural choices in their performance and low-dimensional representations.

References

[Koh82] Teuvo Kohonen. "Self-organized formation of topologically correct feature maps". In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. ISSN: 1432-0770. DOI: 10.1007/bf00337288. URL: http://dx.doi.org/10.1007/BF00337288.

[Fri94] Bernd Fritzke. "A Growing Neural Gas Network Learns Topologies". In: *Advances in Neural Information Processing Systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1994. URL: https://proceedings.neurips.cc/paper_files/paper/1994/file/d56b9fc4b0f1be8871f5e1c40c0067e7-Paper.pdf.

[RMD02] A. Rauber, D. Merkl, and M. Dittenbach. "The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data". In: *IEEE Transactions on Neural Networks* 13.6 (2002), pp. 1331–1341. DOI: 10.1109/TNN.2002.804221.