Universität Hamburg
Department Informatik
Knowledge Technology, WTM

An Introduction to Self-Organizing Maps

Proseminar Paper

Proseminar Artificial Intelligence

Christoph Brauer
Matr.Nr. 6280610
0brauer@informatik.uni-hamburg.de

13.7.2012

Abstract

This paper gives an introduction to Self-Organizing Maps (SOMs), also known as Self-Organizing Feature Maps or Kohonen Maps, as initially presented by Tuevo Kohonen [Koh82].
A brief outline of SOMs is given, the biological motivation of SOMs is pointed out and the given outline is refined to a detailed description accompanied by various examples. Finally a demonstration of how to apply a SOM to the task of image color quantization is presented.

Contents

1 Introduction

1.1 A brief outline

SOMs are meant to model a specific subset of the principles inspired by observations of the human brain's organization process. One might be tempted to think of SOMs as structures simulating real biological behavior, including synapses, neurons, biochemical interactions etc., yet this intuition is misleading [Hay98]. SOMs cannot simulate the complex parallel workings of the human nervous system, but they aim to reproduce an observed behavioral principle and by doing so they in fact do turn out to be powerful tools as we point out in this paper.
The special principle that SOMs aim to imitate is the mapping of high-dimensional input data to a low-dimensional output network in a topology preserving way, i.e. similar inputs are mapped to similar outputs. This feature makes SOMs especially useful for visualization tasks of complex structures that would otherwise be hardly recognizable by humans. Though Self-Organizing Maps form a subset of so-called artificial neural networks [Kri07], no prior knowledge of these is required to fully understand the inner workings of SOMs.

1.2 The biological motivation

One part of the human brain, the visual cortex, gives a descriptive example of the principles how SOMs are organized: The field of view as perceived by sensory cells in the human eye is mapped to the visual cortex such that continuous areas in the view are mapped to continuous planar neural networks in the cortex [Roj96], so similar regions in the input space of the eye are mapped to similar regions in the output space of the brain. By storing and processing high dimensional visual information in the visual cortex, the brain must clearly quantize that input for its planar neural topology.
Even further, the parts of the view that are most important for recognition, i.e. the center parts, are mapped to relative large areas of the cortex though they form a relative small area in the whole field of view, so information density is non-uniform and determined by its importance.
It has as well been observed that the absolute positions of the visual cortex' networks are not fixed, so in topological terms it is not import where information is stored, but it is important in which context information is stored. The details of how SOMs model this biological behavior are discussed in the next chapter of this paper.

2 Fundamentals of Self-Organizing Maps

A Self-Organizing Map is comprised of two building blocks, a node set data structure representing the actual map content and algorithms the apply to that node set. The application of algorithms to the node set forms the actual organization process, and for that organization process modifies nodes according to input vectors, organization has the meaning of learning in biological terms. The basic principle of building a SOM is to set certain operational parameters, initialize its node set and apply its algorithms to modify its nodes set according to the inputs presented. The output of any SOM organization process is just its node set representing the final state of the SOM, so hereby that node set is referenced as *output* node set. For the generic nature of any SOM, the output node set is a generic structure as well, so usually further processing is required to extract the required information. Unlike neural networks which perform supervised learning, SOMs can learn unsupervised, i.e. they do not have to be presented expected results beforehand.

2.1 The output node set

The output node set models the actual *map* of a SOM and represents its internal state. The elements of the output set, referred to as output nodes accordingly, possess two attributes each, a fixed *position* and a mutable *weight*. The position refers to a node's topological location on the map, the weight holds the actual information payload associated with the node. In biological terms the map corresponds to a memory network made up of neurons, which in turn correspond to nodes.

Note that the node set and the map are just different representations of a SOM's node collection, and depending on the context it is either convenient to address nodes as tuples of the output node set in a mathematical fashion or as structures indexed by their positions on a map. For illustrative purposes it is most convenient to display the map representation so one can easily figure out the topological map layout.

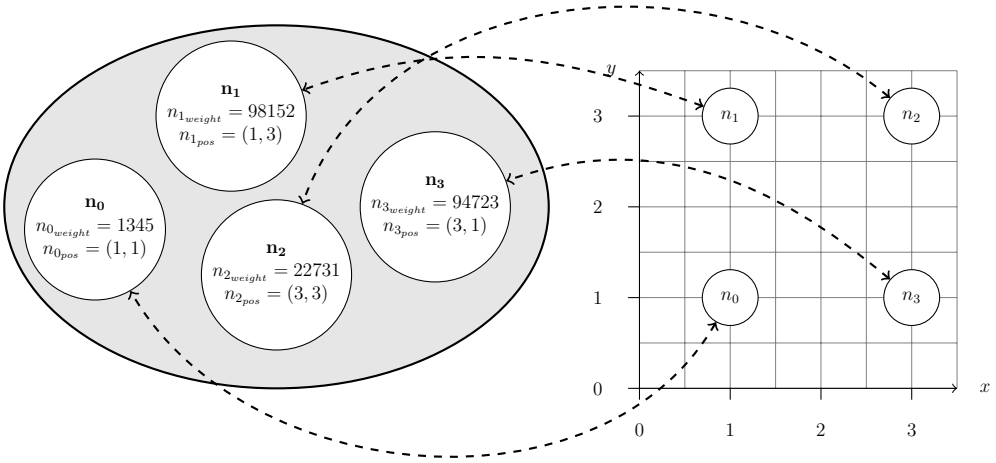

Figure 1: Set and map node representations

Though in practice one- or two-dimensional maps with a flat or rectangular layout as shown in figures 2 and 4 are used most commonly, certain tasks might benefit from more exotic layouts as shown in figures 3, 5, 6 and 7.

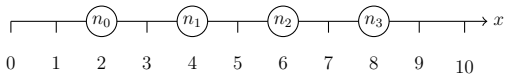

Figure 2: A uniformly spaced one-dimensional layout

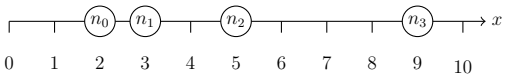

Figure 3: A non-uniformly spaced one-dimensional layout

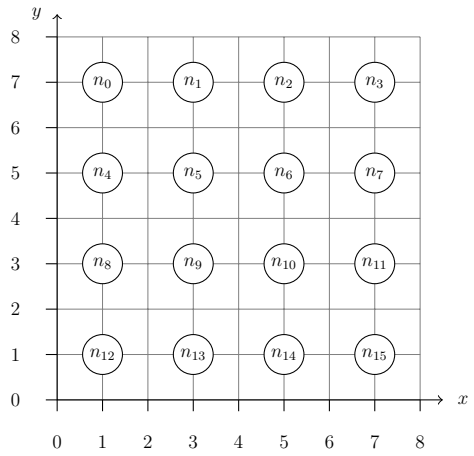

Figure 4: A uniformly spaced two-dimensional rectangular layout

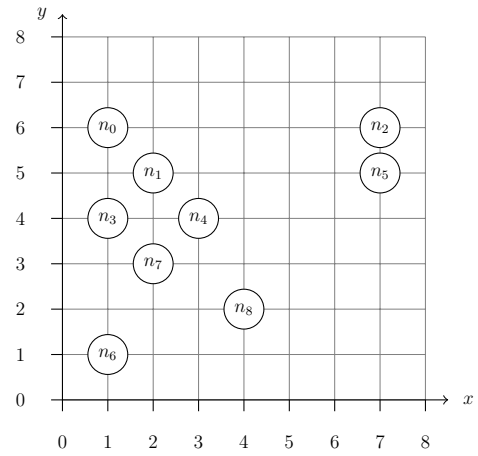

Figure 5: A non-uniformly spaced two-dimensional layout

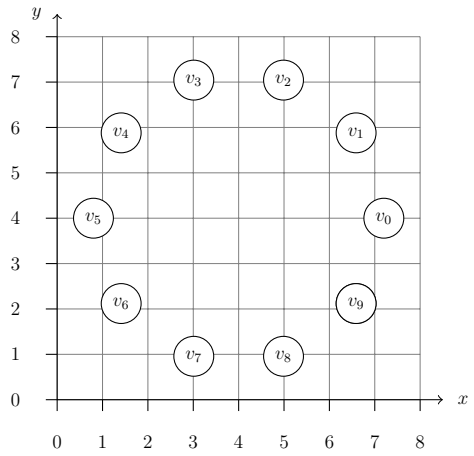

Figure 6: A uniformly spaced two-dimensional circular layout

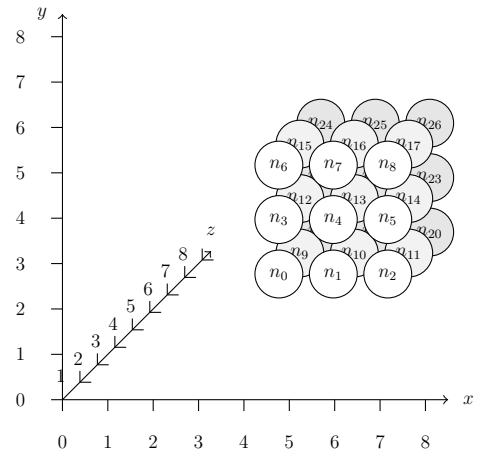

Figure 7: A uniformly spaced three-dimensional cubic layout

Note that the positions of the nodes are initially set up once and not altered by the map organization process afterwards. This might sound a little confusing at a first glance for one might imagine the nodes to "move" for topological preservation, but actually it is never the nodes' positions to be altered but the nodes' weights. This fixed positioning keeps the neighborhood relations between nodes immutable, which in turn is the key for topological preservation.
In the upcoming description of how nodes are organized, variations of topological

positions are denoted as *distances* and variations of weights are denoted as *differences* to clearly distinguish both.

2.2 Creation and Organization outlined

Before going into detail, a simplified brief outline of the SOM creation and organization is given to point out the working principles.

2.2.1 Initialization

Create a fixed-size output node set such that each node is assigned a position according to the desired map layout and a weight having the same dimensionality as the expected input vectors. It is common practice to set the weights to random values, but for certain tasks it might as well be useful to take specific presets.

2.2.2 Organization

The map organization is performed by the application of learning cycles as denoted below:

1.) For each learning cycle do

  2.) For each input vector do

    3.1) Find the single output node which weight matches the input vector most closely. If there are multiple output nodes that equally match the input vector, choose a random one of those.

    3.2) This step forms the heart of the organization process:
    Adjust the weight of the winning output node just found to closer match the input vector, and adjust the weights of this node's topological neighborhood as well.

3 Details of Self-Organizing Maps

3.1 Creation

Let S denote the output node set to be created, and let I denote the input vector space. For each node $n_i \in S$ its position $n_{i_{pos}}$ must be set according to the desired output map layout, and its weight $n_{i_{weight}}$ must be initialized such that $dim(n_{i_{weight}}) = dim(I)$.

Example 1: Output node set creation
An output node set $S := \{n_0, n_1, n_2, n_3\}$ that features a two-dimensional output node map and expects three-dimensional input vectors could be set up as shown in figures 8 and 9.

n_i	$n_{i_{pos}}$	$n_{i_{weight}}$
n_0	(1, 1)	(2, 4, 8)
n_1	(3, 1)	(0, 0, 0)
n_2	(3, 3)	(13, 3, 7)
n_3	(1, 3)	(65, 0, 2)

Figure 8: Node settings

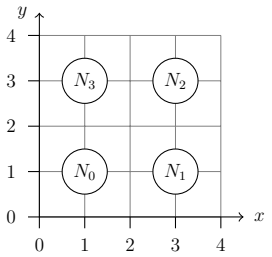

Figure 9: Map layout

Note that the weights were chosen randomly, just any three-dimensional vector would perfectly fit here.

3.2 Organization

In order to specify how the SOM organization is to be performed in detail, according operational parameters must be set beforehand. As outlined in Section 2.2.2, the whole learning process is split into separate learning cycles, so let t_{max} denote the number of these learning cycles. The adaption of nodes' weights to input vectors is affected by another two parameters, the *learning rate* η and the *neighborhood radius* σ. The learning rate specifies to what degree the nodes generally adapt to a given input vector, and the neighborhood radius determines to what degree neighborhood nodes adapt depending on their distance to the winning node. Both parameters are subject to change with time, so let η_0 and σ_0 denote their initial values, $\eta(n)$ and $\sigma(n)$ denote their values according to the time t respectively. Henceforth, let I denote the set of input vectors and let S denote the output node set.

3.2.1 Winning node determination

The winning node is $n_{winner} \in S$ is determined such that among all output nodes it features the least difference between its weight and the input vector v. Hence, n_{winner} must satisfy the condition

$$\forall n_i \in S : \text{diff}(n_{winner_{weight}}, v) \leq \text{diff}(n_{i_{weight}}, v)$$

with $\text{diff}(x, y)$ denoting the difference between x and y. Accordingly, a function $\text{bmu}(v)$ (best matching unit) that identifies the winning node depending on the input vector v can be defined as

$$\text{bmu}(v) = \arg\min \text{diff}(v, n_{i_{weight}}), i = 0, 1, ..., |S| - 1$$

For SOMs can operate on highly complex input vectors such as multimedia data or even text files, no general metric can be applied as a difference function. For the special but common case that the input vector space is just the set $\mathbb{R}^k$, the Euclidean distance is commonly used as a difference function, though even in this case there are multitudes of different metrics that could just be applied as well.

Example 2: Winning node determination
Let S be the node set as denoted in figure 8, and let v be an input vector $v := (10, 20, 30) \in \mathbb{R}^3$. Let the Euclidean distance be chosen as a measure of difference, so for each node $n_i \in S$ the result of the equation $|n_i - v|$ is:

i) $|n_{0_{weight}} - v| = \sqrt{(2-10)^2 + (4-20)^2 + (8-30)^2} \approx 28,35$

ii) $|n_{1_{weight}} - v| = \sqrt{(0-10)^2 + (0-20)^2 + (0-30)^2} \approx 37,42$

iii) $|n_{2_{weight}} - v| = \sqrt{(13-10)^2 + (3-20)^2 + (7-30)^2} \approx 28,76$

iv) $|n_{3_{weight}} - v| = \sqrt{(65-10)^2 + (0-20)^2 + (2-30)^2} \approx 64,88$

The least difference is found between the weight of n_0 and v which makes n_0 the winning node. More formally, n_0 is the winning node for

$$|n_{0_{weight}} - v| = min\{|n_{k_{weight}} - v| : n_k \in S\} \tag{3.1}$$

3.2.2 Weight adaption

The determination of the winning node is followed by the adaption of all output nodes' weights according to the given formula:

$$\underbrace{n_{i_{weight}}(t+1)}_{\text{adapted weight}} := \underbrace{n_{i_{weight}}(t)}_{\text{current weight}} + \underbrace{\underbrace{\eta(t)}_{\text{learning rate}} \cdot \underbrace{h(|n_{winner} - n_i|, t)}_{\text{neighborhood function}}}_{\text{weight scaling}} \cdot \underbrace{(v_i - n_{i_{weight}}(t))}_{\Delta_{\text{weight}}} \tag{3.2}$$

As pointed out in above formula, the degree to what an input node's weight adapts to the input vector is determined by a scaling factor comprised of the current learning rate η and the *neighborhood function* h.

The learning rate η is a real number such that $0 < \eta \leq 1$ and should decrease with time strictly monotonically, hence $\eta(t+1) < \eta(t)$. Popular decay functions

applicable to η are illustrated in figures 10 and 11 with the exponential decay being the most commonly used one.

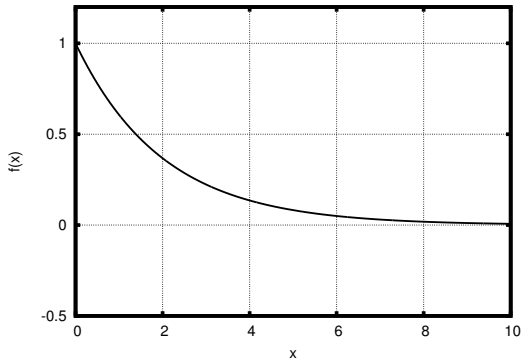

Figure 10: Exponential decay function

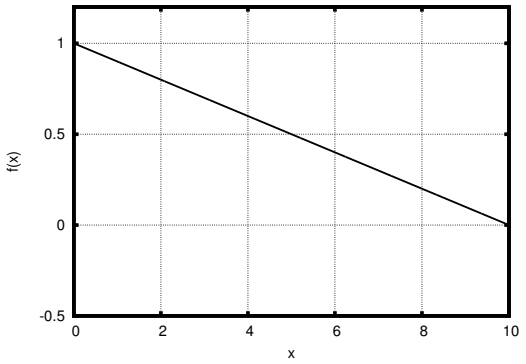

Figure 11: Linear decay function

Equations according to figures 10 and 11 :

$$\text{Exponential decay function: } f(x) = k \cdot e^{-\frac{x}{\lambda}} \text{ for } k = 1, \lambda = 2 \tag{3.3}$$

$$\text{Linear decay function: } f(x) = k + x \cdot m \text{ for } k = 1, m = -\frac{1}{10} \tag{3.4}$$

According to Haykin [Hay98], it is been observed to be a valuable choice letting the learning rate start with an initial value η_0 close to 0.1 gradually decreasing with time towards its final value η_f close to 0.01, though these settings may vary by specific maps. An adaption of the general exponential function as illustrated in figure 10 can be used as a learning rate function such that

$$\eta(t) = \eta_0 \cdot k^{(\frac{t}{t_{max}}) \cdot (log(\frac{\eta_f}{\eta_0}) \cdot \frac{1}{log(k)})} \tag{3.5}$$

for an arbitrary value of $k \neq 1$ as shown in figure 12. Though this function yields the expected results, Haykin proposes a simpler function as denoted by equation (3.6):

$$\eta(t) = \eta_0 \cdot e^{-\frac{t}{t_{max}}} \tag{3.6}$$

This learning rate function reaches its minimum at $\eta(t_{max}) = \eta_0 \cdot \frac{1}{e} \approx \eta_0 \cdot 0.37$ and thus does not exactly match the desired value of 0.01, but it has been observed that in practice this function is sufficient for it decreases towards a value close to 0.01 in an exponential fashion as shown in figure 13.

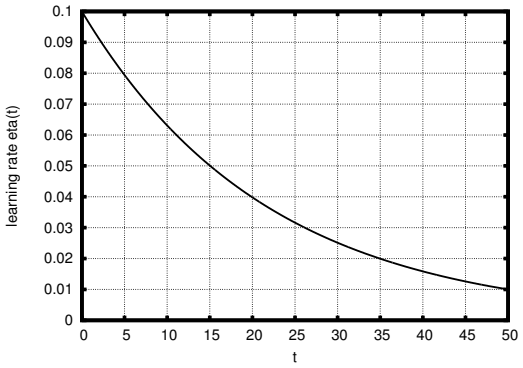

Figure 12: Function graph of eq. (3.5) with $\eta_0 = 0.1$, $\eta_f = 0.01$, $k = 2$ and $t_{max} = 50$

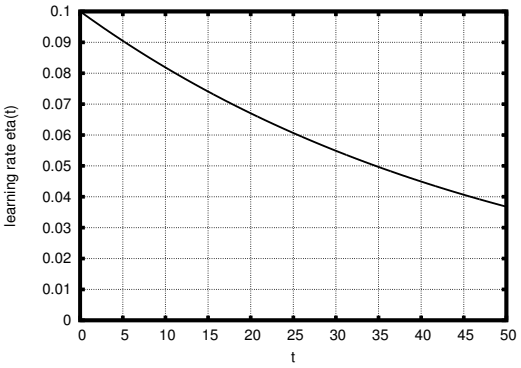

Figure 13: Function graph of eq. (3.6) with $\eta_0 = 0.1$ and $t_{max} = 50$

The neighborhood function specifies to what amount a neighborhood node adapts to the given input vector according to its topological distance from the winning node. One way to imagine the neighborhood function is in terms of gravity: the winning node forms the center of gravity, and the surrounding neighborhood nodes are influenced by the winning node according to the strength of its gravity field denoted by the neighborhood function.

Figures 14, 15, 16 and 17 illustrate normalized popular neighborhood functions with d denoting the distance between the current node and the winning node. The Gaussian function has been observed to yield highly valuable results and thus is the by far most popular choice, the Mexican hat function was proposed by Tuevo Kohonen in his initial SOM paper [Koh82] for it models the biological behavior most closely, and the rectangular and triangular functions form rather simple alternatives.

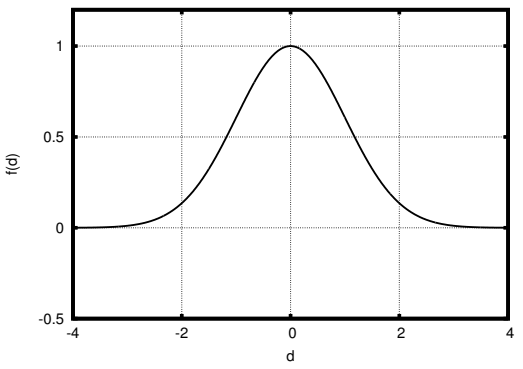

Figure 14: Gaussian function

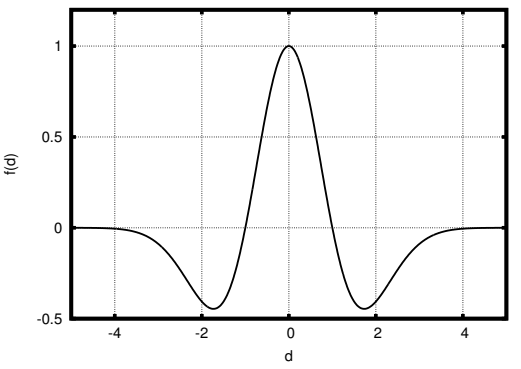

Figure 15: Mexican hat function

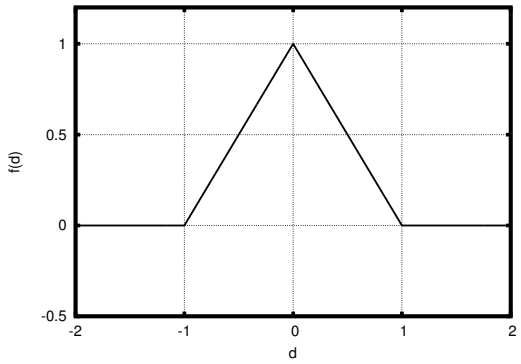

Figure 16: Triangular function

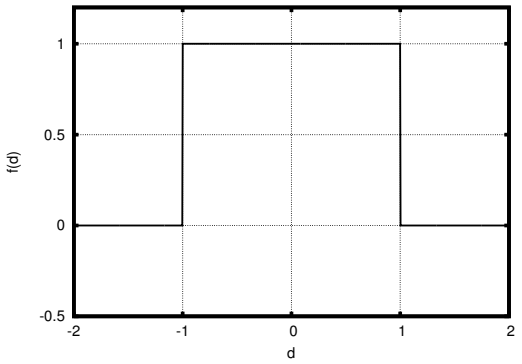

Figure 17: Rectangular function

Equations according to figures 14, 15, 16 and 17 :

$$\text{Gaussian function: } h(d) = e^{-\frac{d^2}{2 \cdot \sigma^2}} \tag{3.7}$$

$$\text{Mexican hat function: } h(d) = (1 - \frac{d^2}{\sigma^2}) \cdot e^{-\frac{d^2}{2 \cdot \sigma^2}} \tag{3.8}$$

$$\text{Triangular function: } h(d) = \begin{cases} 1 - \frac{|d|}{\sigma} & \text{if } |d| \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

$$\text{Rectangular function: } h(d) = \begin{cases} 1 & \text{if } |d| \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

for the neighborhood radius $\sigma = 1$

Note that unlike the difference function which can be specified by various metrics, the topological distance is always determined by the Euclidean distance. As implied by the given example equations, the neighborhood function not only depends on a distance, but on a neighborhood radius σ as well. The meaning of σ depends on which neighborhood function is chosen: for the rectangular or triangular functions, σ denotes the radius in which those functions return a value greater than zero, but for the Gaussian function σ denotes the inflection points, so that function must not simply be assumed to yield zero for a distance greater than σ.

Example 3: Neighborhood function

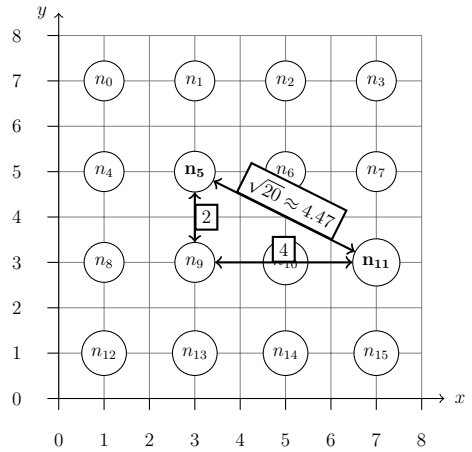

Figure 18: Topological distance $|n_{5_{pos}} - n_{11_{pos}}| \approx 4.47$

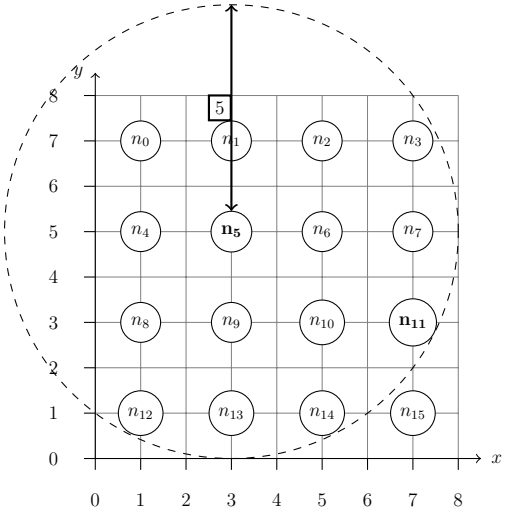

Figure 19: $\sigma = 5.0$ denoted by the dashed circle

Let n_0 denote the winning node and n_{11} denote the current node to be altered. For $\sigma = 5$ the result of the Gaussian neighborhood function $h(d)$ is

$$h(|n_{5_{pos}} - n_{11_{pos}}|) = e^{-\frac{(|n_{5_{pos}} - n_{11_{pos}}|)^2}{2 \cdot 5^2}} = e^{-\frac{20}{50}} = e^{-0.4} \approx 0.67$$

For the sake of simplicity the detail that σ is about to decrease with time has been omitted in the above explanation. To catch up on this property, σ is substituted by $\sigma(t)$, so h depends on time as well and $h(d)$ simply becomes $h(d, t)$. For instance, the Gaussian neighborhood function depending on distance and time is denoted by eq. (3.11):

$$h(d, t) = e^{-\frac{d^2}{2 \cdot \sigma(t)^2}} \tag{3.11}$$

Just like η, σ should shrink gradually by time, either in a linear or an exponential fashion. As proposed by Haykin, the initial neighborhood radius σ_0 should be set to span the whole output map and $\sigma(t)$ should decrease with time such that finally only the winning node itself or its nearest neighbors are included. The according equation is denoted by eq. (3.12) and illustrated in figure 20.

$$\sigma(t) = \sigma_0 \cdot e^{(-\frac{t \cdot \log \sigma_0}{t_{max}})} \tag{3.12}$$

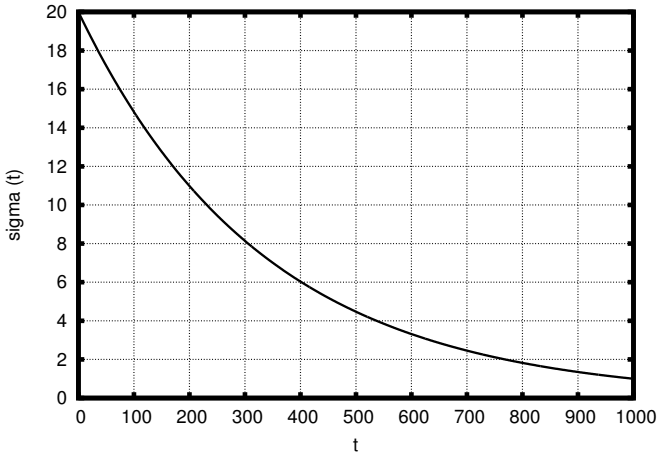

Figure 20: $\sigma(t)$ for $\sigma_0 = 20$ and $t_{max} = 1000$

3.2.3 Complete algorithm

Now that all building blocks of the organization process are introduced, the final algorithm can be composed in an easy understandable bottom-up manner:

---

for $t = 0$ **to** $t_{max} - 1$ **do**
  for each v **in** I **do**
    $n_{winner} \leftarrow \text{bmu}(v)$
    for each n_i **in** S **do**
      $n_{i_{weight}} \leftarrow n_{i_{weight}} + \eta(t) \cdot h(|n_{i_{pos}} - n_{winner_{pos}}|, t) \cdot (v - n_{i_{weight}})$
    end for
  end for
end for
return (S)

---

4 Example - Image Color Quantization

Image color quantization refers to the task of reducing the total amount of distinct colors used to reproduce a digital image. The quality of any color quantization is determined by the similarity of the full-color input image and the reduced-color output image. Though there are certain definitions of "similarity", we refer to similarity as perceived by the human eye when looking at images.
Self-Organizing Maps featuring the ability to group similar data are highly valuable tools for this task. The basic idea of how to use a SOM for image color quantization is to construct a SOM with its output nodes reflecting the desired output colors and passing the pixels of the input image as input vectors to that SOM.
A small Python program has been developed by the author of this paper to implement an SOM based image color quantizer. Without going into too much technical detail here, the implementation follows the upcoming algorithm :

1. Create an empty input vector set I

2. For each 24-bit pixel in the truecolor input image add a (R, G, B) tuple representing that pixel to I

3. Create a SOM with a node set S such that $|S|$ denotes the amount of distinct output colors and each node's weight features 3 dimensions

4. Set the SOM's operational parameters and organize it by presenting each $v \in I$ to it

5. Once the organization is finished, the weights of the output node set represent the colors found, i.e. the new color palette

6. Create the output image such that each pixel of the input image is mapped to the palette color closest to it

4.1 Results

Two 24bit truecolor images were chosen as illustrative sample inputs, a picture of Selene as shown in figure 21 and a picture of Lord Vader, Lando Calrissian and Boba Fett as shown in figure 22. For each picture, the number of colors has been reduced from 16.8M to 16, once using the sample SOM implementation and once using gimp's color reduction function which is known to use the popular median-cut algorithm [Hec82]. Though in terms of speed gimp performed magnitudes faster than the reference SOM implementation, in terms of perceptual quality the results of the SOM implementation can compete or even outperform gimp as illustrated. For a highly optimized implementation of a SOM color quantizer, the program "pngnq" [Coy] based on the NeuQuant algorithm [Dek94] is recommended.

Figure 21: Selene
16.8M colors

Figure 22: Lord Vader et al.
16.8M colors

Figure 23: Selene
16 colors (SOM)

Figure 24: Lord Vader et al.
16 colors (SOM)

Figure 25: Selene
16 colors (gimp)

Figure 26: Lord Vader et al.
16 colors (gimp)

5 Conclusion

Self-Organizing Maps can be valuable tools for a broad range of applications that benefit from the ordering of input data by degrees of similarity. Their generic structure allows for operation on nearly any kind of input data as long as a metric on those can be provided, and their ability to learn in an unsupervised fashion enables them to adapt to even completely unforeseen input patterns without any supervisory intervention.
These advantages make the Self-Organizing Maps popular choices for tasks like clusterization [BLP05], image compression[ALV03], image color quantization[Dek94] or even organization of text document collections [HKLK97].

References

[ALV03] C. Amerijckx, J.-D. Legat, and M. Verleysen. Image compression using self-organizing maps. *Syst. Anal. Model. Simul.*, 43(11):1529–1543, November 2003.

[BLP05] Fernando Bação, Victor Sousa Lobo, and Marco Painho. Self-organizing maps as substitutes for k-means clustering. In Vaidy S. Sunderam, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *International Conference on Computational Science (3)*, volume 3516 of *Lecture Notes in Computer Science*, pages 476–483. Springer, 2005.

[Coy] Stuart Coyle. pngnq. `http://pngnq.sourceforge.net`.

[Dek94] Anthony Dekker. Kohonen neural networks for optimal colour quantization. *Network: Computation in Neural Systems*, 5:351–367, 1994.

[Hay98] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[Hec82] P. S. Heckbert. Color Image Quantization for Frame Buffer Display. *ACM Computer Graphics (ACM SIGGRAPH '82 Proceedings)*, 16(3):297–307, 1982.

[HKLK97] Timo Honkela, Samuel Kaski, Krista Lagus, and Teuvo Kohonen. Websom - self-organizing maps of document collections. In *Neurocomputing*, pages 101–117, 1997.

[Koh82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 10.1007/BF00337288.

[Kri07] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. available at http://www.dkriesel.com.

[Roj96] Raúl Rojas. *Neural networks: a systematic introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.