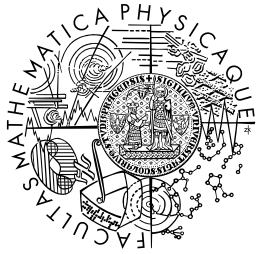


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Kyung Won Park

Extending Self-organizing Maps with Ranking Awareness

Department of Software Engineering

Supervisor of the bachelor thesis: Mgr. Ladislav Peška, Ph.D.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, Mgr. Ladislav Peška, Ph.D. for his kind and patient guidance throughout this work. I would also like to express my gratitude to RNDr. Miroslav Kratochvíl, Ph.D. for all his ideas and suggestions.

Lastly, I dedicate this work to my friends and family. None of this would be possible without you.

Title: Extending Self-organizing Maps with Ranking Awareness

Author: Kyung Won Park

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., Department of Software Engineering

Abstract: The self-organizing map (SOM) is a powerful clustering algorithm which takes high-dimensional data as the input and produces a low-dimensional representation of the data. The SOM provides useful insights into the given data by recognizing similar input vectors and clustering them. However, they take into account only the local similarity of the input data, as opposed to relevance (any external ranking). In this paper, we propose two ranking-aware variants of the SOM in an effort to tackle this issue and incorporate evaluation metrics to evaluate our results.

Keywords: self-organizing map, relevence feedback, known-item search

Contents

Introduction **3**

1 Preliminaries **5**
1.1 Information retrieval . . . 5
1.1.1 Known-item search . . . 5
1.1.2 Evaluation of an IR system . . . 5
1.1.3 Exploration-exploitation dilemma . . . 7
1.2 Dataset overview . . . 7
1.3 Machine learning . . . 7
1.3.1 Process . . . 8
1.3.2 Approaches . . . 9
1.4 Artificial neural networks . . . 11

2 Self-Organizing Map **12**
2.1 Overview . . . 12
2.2 Training algorithm . . . 14
2.2.1 Initialization . . . 14
2.2.2 Competition . . . 14
2.2.3 Cooperation . . . 14
2.2.4 Adaptation . . . 16
2.2.5 Online SOM algorithm . . . 18
2.2.6 Batch SOM algorithm . . . 18
2.3 Evaluation methods . . . 19
2.3.1 Quantization error . . . 20
2.3.2 Topographic error . . . 20
2.3.3 U-matrix . . . 20

3 Experiments **22**
3.1 Aim . . . 22
3.2 Rating dimension SOM . . . 24
3.2.1 Initial bias . . . 24

3.2.2 Re-biasing 25
3.3 Rating-weighted SOM 27
3.4 Evaluation protocol 28

4 Results and Discussion **31**

Conclusion **39**

Bibliography **41**

List of Figures **43**

List of Acronyms **44**

Introduction

Motivation

The self-organizing map (SOM) is an artificial neural network which preserves the topology of the input data by producing a low-dimensional (typically 2-dimensional) representation of the data. In this work, we consider interactive video retrieval task in which the user provides a query describing their search needs; in our case, a target image. The SOM takes as input images represented by feature vectors, and produces a grid (whose size is specified by the user) of images as output in which similar images are mapped to adjacent nodes. This poses a problem to our task: the SOM considers only the local similarity of the input data as it lacks the capability to take into account relevance (any external ranking) even if it exists. In this work, we will experiment some of the possible approaches to tackle this issue and evaluate each of them based on criteria which will be mentioned in this section shortly.

Contrary to the SOM-based approach which solely considers local similarity, one could take an approach which selects and displays images solely based on their relevance scores (so that only K highest scoring images are displayed on a grid of K nodes). Such approach is called Top-K display. Since Top-K display only takes into consideration an external ranking of given data, it does not preserve the topology of the original data. Similarly, it is clear that the SOM-based approach does not preserve any external ranking of given data. These two approaches are described by a more general and well-known problem: the exploration-exploitation dilemma [1], which describes the trade-off between exploration and exploitation. Exploration means making a decision that appears to be non-optimal at the moment, with hopes that the data observed thus far is not sufficient to truly determine the optimal decision (which in turn better preserves the topology and diversity of given data). On the other hand, exploitation means making the optimal decision based on the data observed thus far. In our case, exploration corresponds to the SOM-based approach whereas exploitation corresponds to Top-K display. In this work, we aim to find a middle ground between the two which preserves both local similarity and relevance of given data.

Related work

Several others published works aimed at incorporating the notion of relevance in the SOM, such as SOMHunter, Dimension-selective SOM (DSSOM) and Rating-aware SOM [2, 3, 4]. SOMHunter takes

into consideration relevance of items with respect to a given query, resulting with SOM with regions of highly relevant items [2]. However, one drawback is that the items within said regions don't preserve the topology of the original data. Furthermore, the position of items within the regions is arbitrary [4]. More recently, Peška and Lokoč devised Rating-aware SOM by optimizing the SOM with respect to non-discounted cumulative gain (nDCG) [4]. Rating-aware SOM explored one of possible approaches to modifying the SOM; more specifically, its best matching unit (BMU) selection procedure applied during training.

Peška and Lokoč delved into two of the possible node ordering for defining the "most prominent area of the display" [4]. In this work, we will consider the row-first ordering, in which upper rows are assumed to be more prominent than lower rows.

Aim

Our aim is to devise ranking-aware SOM variants by integrating the exploration-based approach of the original SOM with the exploitation-based approach (Top-K display). We will evaluate our variants with several evaluation metrics we devised, and determine whether (and to what extent) they preserve the benefits of both approaches.

Thesis overview

This thesis is divided into four chapters.

In Chapter 1, background information for the rest of our paper will be introduced, such as information retrieval, normalized discounted cumulative gain, and machine learning.

In Chapter 2, the original SOM will be described in detail.

In Chapter 3, we propose two ranking-aware SOM variants and their sub-variants. We also describe the evaluation metrics to be used.

In Chapter 4, we evaluate our variants and examine their displays in more detail, using example target images.

Chapter 1

Preliminaries

This chapter aims to provide background information needed to understand our goal and experiments. The self-organizing map will be described in detail in Chapter 2.

1.1 Information retrieval

Information retrieval (IR) is the process of searching, locating and retrieving information that is relevant to an information need. Such need is identified by a user query. A query often takes the form of a text string, to which an IR system returns a set of relevant items. An IR system does this by assigning a numeric score called relevance score to each item in the database. Results generated by IR searching are typically ranked by these scores and displayed in the order of relevance. In order to improve the result set of an IR system for a given query, it is common to take an initial set of results and gather user feedback on whether or not individual items in the set were relevant. This is called relevance feedback.

1.1.1 Known-item search

Known-item search (KIS) refers to a task in which the user searches for a particular item. KIS is a special case of IR, in which the user typically describes their information needs for some concrete item with a query. For example, the user may wish to search for a particular scene occurring in a video on YouTube by typing a text query describing that scene.

For KIS to be efficient, it is important that user queries are accurate (they provide an accurate description of the sought item) and the query-processing algorithm works correctly.

1.1.2 Evaluation of an IR system

Evaluating an IR system is quite a complicated task. It involves measuring how well the information retrieved by the system satisfies the information needs of the users. However, this can vary because

different users may consider different items in the result to be "relevant". Nonetheless, we can use evaluation metrics to quantify the performance of an IR system as well as objectively compare the performances of distinct IR systems.

In this work, we evaluate the performance of the Self-organizing map (SOM) as an IR system. As IR systems typically return a ranked list of items based on their degree of relevance, We want highly relevant items to appear at the top of the ranking rather than mildly relevant or irrelevant items, and vice versa. We use normalized discounted cumulative gain (nDCG) to measure the goodness of our new ranking induced by the SOM in comparison to the original ranking.

Normalized discounted cumulative gain

The notion of normalized discounted cumulative gain (nDCG) was derived from cumulative gain (CG) and discounted cumulative gain (DCG). Very often, IR systems assign some sort of relevance scores to items in a list and rank them based on their relevant scores. CG is the sum of the relevance scores in the list. CG alone does not provide much information about ranking quality; all items in the ranking carry equal importance regardless of their individual ranks. CG may be identical for two vastly different rankings since it is simply the sum of all relevance scores in the list. DCG resolves this issue by penalizing items with high relevant scores appearing at lower ranks. DCG of an item at a particular rank position p is calculated as follows:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where rel_i denotes the relevance score of the i-th item. $\log_2(i+1)$ ensures that items at lower ranks contribute less towards the total DCG value. While the DCG is a significant improvement from CG on measuring rank correlation (similarity of two rankings), it may provide unreliable results in case the rankings have varying sizes (p value in the DCG formula). For example, a ranking of 50 items and a ranking of 5 items would be difficult to compare with the DCG since it's highly likely that the former would yield a greater DCG value. Furthermore, it'd benefit us to normalize the DCG and have it take on a value in a certain range. We do this by sorting the items in the ranking by relavance score, taking top p items and obtaining the maximum possible DCG, called ideal DCG (IDCG). Then we obtain the nDCG as follows:

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

The nDCG takes on a value in the range [0, 1], where the perfect ranking (one in which DCG = IDCG) yields an nDCG of 1. Figure 1.1 illustrates an example of nDCG calculation with two example rankings: one which is well-ordered, and one which is not. For the first ranking in particular, we can observe that DCG = IDCG, resulting in nDCG = 1.

i	rel_i	$2^{rel_i} - 1$	$\log_2(i+1)$	$2^{rel_i} - 1 / \log_2(i+1)$
1	1.0	1	1	1
2	0.8	0.741	1.585	0.468
3	0.7	0.625	2	0.313
4	0.5	0.414	2.322	0.178
5	0.2	0.149	2.585	0.058

$DCG_5 = 2.14$
$nDCG_5 = 1.0$

j	rel_j	$2^{rel_j} - 1$	$\log_2(j+1)$	$2^{rel_j} - 1 / \log_2(j+1)$
1	0.1	0.071	1	0.071
2	0.2	0.149	1.585	0.094
3	0.3	0.231	2	0.116
4	1.0	1	2.322	0.431
5	0.5	0.414	2.585	0.160

$DCG_5 = 1.015$
$nDCG_5 = 0.638$

Figure 1.1: Example of nDCG calculation

1.1.3 Exploration-exploitation dilemma

The exploration-exploitation dilemma is a well-known problem in IR, which describes the trade-off between exploration and exploitation. In IR systems, there's always a choice of what items to display to the user: do we (only) display items that are highly relevant to the given query, or do we display items that are less relevant, but representative of the entire dataset? If the system opts for the "safe" choice and presents items that are most relevant to the user at a given moment (exploitation), it loses the possibility to gather information on other information, which can possibly be better (more relevant) than the current ones. On the other hand, if the system presents various items regardless of the degree of relevance (exploration), it may not provide good results to the user. Evidently, find a good middle ground between exploration and exploitation is an important goal for IR systems.

1.2 Dataset overview

We used the V3C1 dataset [5] of 20,000 selected video frames, whose features were extracted using convolutional neural networks. As described in detail by Peška and Lokoč [4], for a target image, the relevance score for each item is calculated based on the Euclidean distance with some noise added to simulate a real-life KIS scenario.

1.3 Machine learning

Machine Learning has been arguably one of the most influential technologies in the 21st century. It learns from large amounts of data by recognizing patterns from the given data, and applies this information to new data that it has not seen before. There are infinitely many applications of machine

learning, from credit-card fraud detection and machine translation to marketing. A formal definition of machine learning was given in 1997 by Tom Mitchel: a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E [6].

1.3.1 Process

In machine learning, a common task is to train the machine learning model on data, and make predictions on data it hasn't seen before. Building a machine learning model is an iterative process. Generally speaking, the process can be divided into 5 steps: data collection, data preprocessing and exploration, training, evaluation, and parameter tuning.

Data collection

As the name suggests, this step involves collecting data to train our machine learning model on. This step, along with the next step, are crucial because building an accurate model heavily relies on the quantity and quality of data used. It's important that the data is accurate and meaningful for our purpose.

Data comes in several forms: structured or unstructured. Structured data refers to data that is stored in a machine-readable and organized format. Structured data is usually stored in a table, with rows and columns. Common examples of structured data include spreadsheets and SQL databases. Unstructured data is the opposite; it has no predefined structure, and therefore is difficult to process for the machine. Structured data is much easier to train a machine learning model on and thus used to train machine learning models most of the time.

Data preprocessing and exploration

During this step, we explore the data to gain a better insight, and format it into an optimal format for training. Usually the first task performed during the step is data cleaning: in case our raw data contains corrupt, inaccurate or irrelevant data, we either modify it or delete it. Afterwards, we explore the data via statistical analysis and visualization. Statistical analysis helps us understand the data better by providing information such as the distribution of each variable or relationship between different variables. Likewise, We may plot the dataset or features against each other in order to identify potential patterns.

Training

During the training process, the model uses the data to incrementally improve its ability to make predictions. This is achieved by adjusting parameters of the model to better represent, or "fit" the training data. Let us take Figure 1.2 as an example. At the very beginning of the training, the line drawn by the model may represent the data quite poorly. Through training, it adjusts its m and b

parameters in $y = mx + b$, resulting in the line better fitting the data points. The goodness of the fit is measured by the cost function, which in our example is usually the least squares:

$$S = \sum_{i=1}^{n} r_i^2$$

where $r_i = y_i - f(x_i)$.

Evaluation

Once training is complete, the model is evaluated using the test set. This is intended to be representative of how the model may perform in a real world scenario. Notice how the quality and quantity of the data we select plays a pivotal role in the model's performance. Usually, various evaluation metrics are available for evaluating the machine learning model of choice.

Parameter tuning

During this step, the model tries to see if it can further improve its performance by tuning the parameters. These parameters are often referred to as hyperparameters because their values are used to control the learning process, as opposed to other parameters which are "learned" by the model.

1.3.2 Approaches

Machine learning algorithms are divided into two main categories based on the learning approach they take: supervised learning and unsupervised learning

Supervised learning

In supervised learning, all of the input data is labelled; the data is given in the form of input-output pairs, often denoted (x_i, y_i) for $i = 1, 2, ..., n$ where n is the number of data points. x_i is called the input variable, and y_i is called the label or target variable. The goal is to find the pattern between the input variable and the target variable. Alternatively, the goal is to learn the mapping that describes the relationship between the two. If the target variable is a categorical variable (one which takes on a finite number of possible values), then we are dealing with a classification problem. If the target variable is continuous, then we speak of a regression problem.

Classification

In classification, the machine learning model that we train is called a classifier. We train the classifier to predict the correct output class of our input. Since the number of possible values for the target variable is finite, trained classifiers usually assign a probability to each output class based on the input and select the class with the highest probability to be the predicted output. A good example

of classification is image classification. The task is to predict a single label for a given image. For instance, let us suppose we have an animal species classification task in which the possible labels are "dog" and "cat". Since there are only 2 possible labels, the task is called a binary classification task. Given an input image, our classifier will predict whether the image contains a dog or cat.

Regression

In regression, the goal is to predict the value of the target variable based on a given input variable. Since the target variable is continuous, it is practically impossible to accurately predict the true value. Therefore, the focus is on making as close a prediction as possible with respect to the true value. A common example of regression is linear regression. The simplest form of linear regression is when the input variable and target variable are both one-dimensional. The machine learning model will attempt to find a linear relationship between the two variables. The goal is to find a line that best fits the input data.

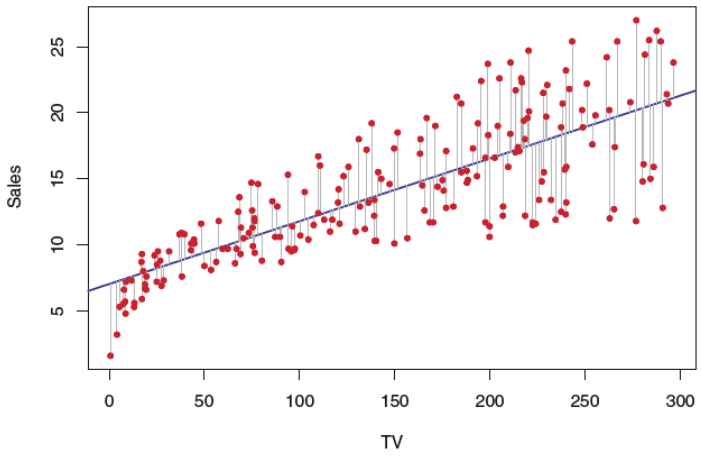
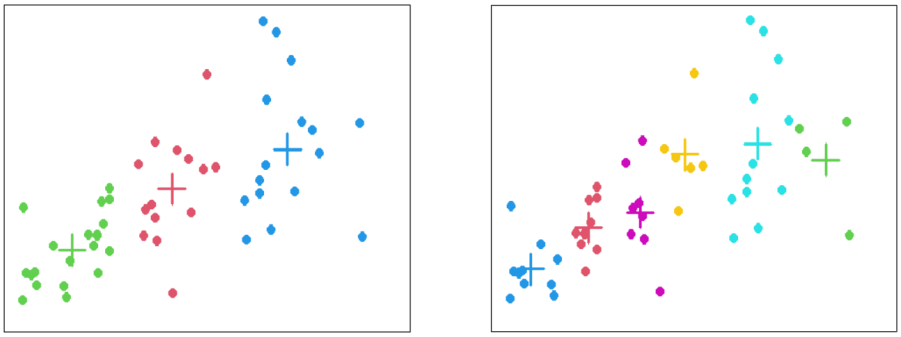


Figure 1.2: Example of linear regression. Source: [7]

Unsupervised learning

Unlike supervised learning, data used in unsupervised learning has no label. In other words, for every x_i in $i = 1, 2, ..., n$, corresponding y_i does not exist. This means the computer has to learn from the given input, without guidance. Naturally, the goal in unsupervised learning is to detect underlying patterns and structure in the data. A common usage of unsupervised learning is clustering. Clustering is the task of partitioning data into groups such that items in the same group are similar to each other in comparison to those in other groups. Each such group is called a cluster. Clustering can help us explore, analyze and better understand our data. A common example of a clustering algorithm is k-means clustering. With the value of k specified by the user, the algorithm divides the data into k clusters as demonstrated below:



(a) k = 3 (b) k = 6

Figure 1.3: Example of k-means clustering with different k values. The crosses represent the centroids of each cluster.

1.4 Artificial neural networks

This section will only cover the basics of artificial neural networks needed to understand the SOM. An artificial neural network (ANN) is an algorithm that is inspired by how neurons in the human brain function. There are typically three types of layers in a typical ANN architecture: the input layer, the output layer, and the hidden layer(s) in between the two aforementioned layers. Each layer consists of one or more nodes (or neurons), which are connected only to other nodes in the immediately preceding and following layers. Each connection between nodes has a weight, which determines the degree of influence one node has on the other. A very important characteristic of these weights is that they can be trained. In other words, the algorithm "reads" the input differently at each iteration in order to learn and find the optimal weights for the given data.

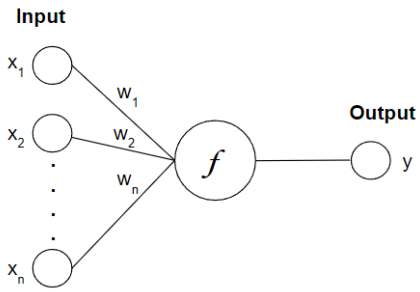


Figure 1.4: A very simple ANN. f denotes some function which processes each input node with its weight and produces an output y.

Chapter 2

Self-Organizing Map

The self-organizing map (SOM) is an artificial neural network which was introduced in 1982 by Teuvo Kohonen [8]. The SOM is trained using unsupervised learning, and is a powerful clustering algorithm often used to simplify high dimensional input data into a low-dimensional (typically 2-dimensional), topologically preserved representation.

2.1 Overview

The SOM was inspired by how our brain processes sensory information. Such information is mapped to distinct parts of the cerebral cortex in such a way that neurons containing similar information are physically close to each other and can exchange signals. However, the SOM was not the first attempt in imitating our brain's cognitive function; In 1973, Von der Malsburg [9] invented earlier models of self-organizing neural networks. Malsburg's model had a crucial downfall that their self-organizing power was quite weak. Furthermore, it required the input and output layer be of the same dimension, which meant that the model had limited usage. Based on these predecessors, Kohonen later invented the SOM which compresses input data (regardless of its dimension) into a lower (often two-dimensional) output with a learning algorithm that is simpler and more intuitive. The SOM consists of a set of nodes, which are fully connected to the input layer and often arranged in a 2D rectangular grid. The nodes in the SOM are not connected to one another. Given an input space of dimension n and the total number of nodes m specified by the user, each node in the network is associated with a n-dimensional weight vector $w_i, i \in= \{1, 2, ..., m\}$. Each weight vector represents the position of the corresponding node in the input space. In training, the pairwise distances between the input x and $w_1, w_2, ..., w_m$ are calculated. The training process moves the weight vectors towards the input data, resulting in the SOM eventually preserving the topology of the input space.

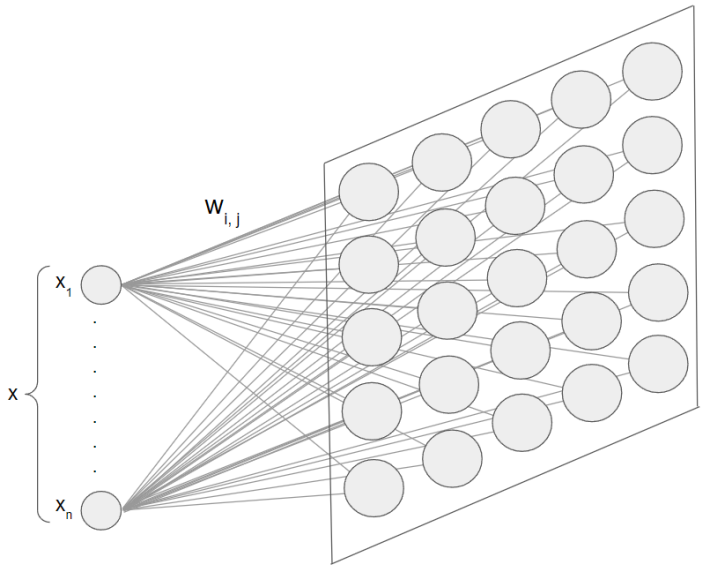


Figure 2.1: Structure of SOM. x denotes an input vector with n dimensions, and the grid on the right depicts the SOM. The lines in the middle depict weight vectors associated with each SOM node, connected to each dimension of x.

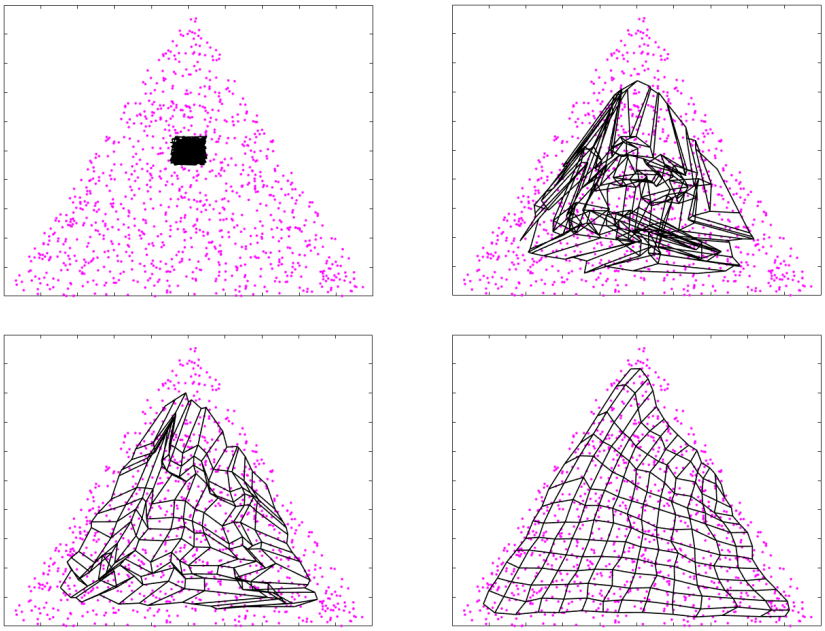


Figure 2.2: Example usage of SOM demonstrating SOM's topology preservation, in clockwise direction from top left. Source: [10]

2.2 Training algorithm

There exist two widely used training algorithms of the SOM today: the online SOM algorithm and the batch SOM algorithm. The former is the algorithm introduced in Kohonen's original paper, and the latter was introduced also by Kohonen in 1995. The key difference is that the online SOM algorithm considers a single input vector for each iteration and the batch SOM uses the entire dataset. The batch SOM algorithm is usually preferred in practice over the predecessor, due to its faster computation speed and having fewer learning parameters [11].

The training process for the SOM can be divided into four steps: initialization, competition, cooperation, and adaptation. We will discuss these steps first and delve into where the two algorithms differ.

2.2.1 Initialization

Prior to training, each weight vector must be initialized. Given the total number of SOM nodes m as the input, weight vectors $w_1, w_2, ..., w_m$ are initialized. This can be done in several ways:

1. Random initialization. Weights are assigned from samples from a probability distribution, typically the standard normal distribution. This is the most commonly used method.

2. Linear initialization. Weights are initialized to span the first two principal components. It is known to make the SOM converge faster [12].

3. Sample initialization. Weights are assigned to random samples from an input dataset.

2.2.2 Competition

During this step, the SOM nodes compete to be the winning neuron, often called the best matching unit (BMU) in the context of SOM. Let $x \in \mathbb{R}^n$ be an input vector. The SOM calculates pairwise distances between x and $w_1, w_2, ...$ and selects the closest node from $\boldsymbol{x}$ (i.e. node with the smallest distance) as the BMU. Several distance metrics have been used in the SOM, but the Euclidean distance is by far the most popular choice. Let $bmu(x)$ be the index of the BMU for x. Then $bmu(x)$ is selected by:

$$bmu(x) = \underset{i}{\operatorname{argmin}} \|x - w_i\|, i = 1, 2, ..., m \tag{2.1}$$

2.2.3 Cooperation

A core property of the SOM is that similar input items are grouped towards one area in the output space. In other words, adjacent SOM nodes represent similar items from the input. This is done during the cooperation process. Now that the BMU has been selected for x, the next step is to calculate which other nodes are within the neighborhood of $bmu(x)$. The SOM alters not only the BMU, but the nodes that are "sufficiently adjacent". This is determined by the neighborhood function, for which there are a few choices. Among them, the Gaussian function is the most popular choice:

$$h_{i,bmu(x)} = e^{-\frac{d_{i,x}^2}{2\sigma^2}}$$

It should be noted that Gaussian function's neighborhood is gradual.
where $d_{i,x}^2 = \|\boldsymbol{r_i} - \boldsymbol{r_x}\|^2$ and $\boldsymbol{r_i}$ is the position of node i. $h_{i,bmu(x)}$ denotes the neighborhood distance between the BMU and another node i. Note that for either choice of functions, $h_{i,bmu(x)}$ decreases as $d_{i,x}$ increases and $h_{i,bmu(x)}$ is at its maximum when $d_{i,x} = 0$. There are two parameters from the above formulas yet to be mentioned: σ and t, which entail understanding an important characteristic of the SOM: its learning slows down over time and the map eventually converges. Likewise, the radius of the neighborhood shrinks with time, eventually shrinking to the size of one node, itself. This is accomplished by σ, which is typically the exponential decay function:

$$\sigma(t) = \sigma(t_0)e^{\frac{-t}{\lambda}}, t = 1, 2, 3, ..., k. \tag{2.2}$$

where σ_0 denotes the initial neighborhood radius, λ is the time constant used to decay the learning rate and neighborhood radius, and t is the current time-step of the algorithm (which keeps track of time during the learning process). Summing up, the Gaussian function as our neighborhood function takes the following form:

$$h_{i,bmu(x)}(t) = e^{-\frac{d_{i,x}^2}{2\sigma(t)^2}} \tag{2.3}$$

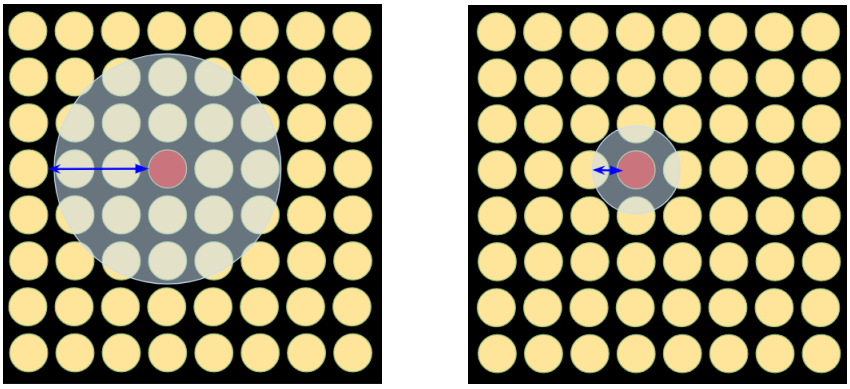


Figure 2.3: Illustration of neighborhood radius decay over time during training. As the radius decreases, fewer nodes are influenced by the change caused by the BMU.

2.2.4 Adaptation

During the adaptation process, the weight vectors for nodes within the BMU's neighborhood are updated as:

$$w_i(t+1) = w_i(t) + \alpha(t)h_{i,bmu(x)}(t)[x - w_i(t)] \tag{2.4}$$

where α is the monotonically decreasing learning rate parameter in the range [0, 1], which is usually the exponential decay function σ:

$$\alpha(t) = \alpha(t_0)e^{\frac{-t}{\lambda}}, t = 1, 2, 3, ..., k. \tag{2.5}$$

That is, the "updating rate" for a weight vector is determined by the current learning rate and neighborhood function value for the corresponding node. The further the node is located from the BMU and the later in the learning process, the lower the updating rate gets.

Once we repeat this training process for all input items for a given number of iterations, the training is complete.

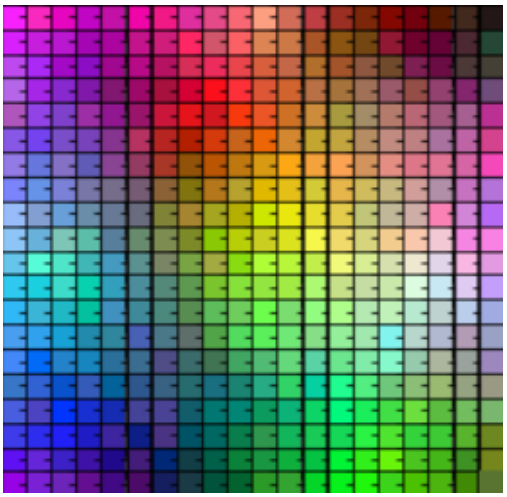


Figure 2.4: Example usage of SOM

Figure 2.4 demonstrates an example usage of the SOM; we can observe that the SOM groups similar colors together. We use 1,000 randomly generated colors represented by RGB vectors (three-dimensional vectors with corresponding RGB values following a uniform distribution in the range [0, 1]) to train a 20-by-20 SOM. After training, we determine the BMU of each input item to display the output. Since the number of input items is much greater than the total number of nodes in this example, it's likely that multiple images are assigned to the same node. In such a case, we selected an arbitrary image.

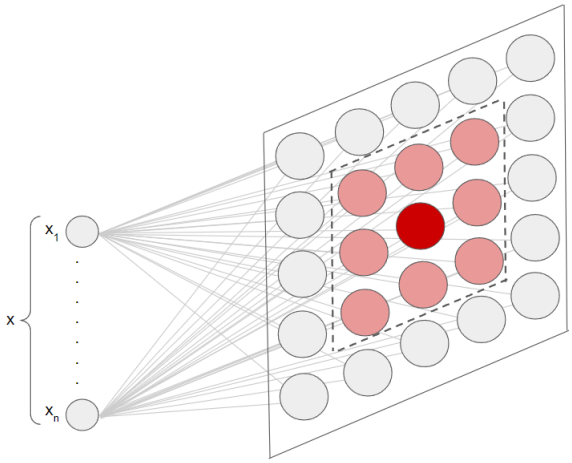


Figure 2.5: BMU (the node in the center of the SOM) and nodes within its neighborhood radius

Summary

Let us summarize the variables and parameters we use for the SOM:

- x is the input vector. X denotes the entire dataset.
- n is the input length (the dimension of x).
- m is the total number of SOM nodes.
- $w_1, w_2, ..., w_m$ are weight vectors for the corresponding SOM nodes.
- t is the current iteration of the algorithm.
- k is the total number of iterations, specified by the user.
- $bmu(x)$ is the BMU of x.
- λ is the time constant.
- $\alpha(t)$ is the learning rate for t given by:
$$\alpha(t) = \alpha(t_0)e^{\frac{-t}{\lambda}}, t = 1, 2, 3, ..., k.$$
- $\sigma(t)$ is the neighborhood radius for t given by:
$$\sigma(t) = \sigma(t_0)e^{\frac{-t}{\lambda}}, t = 1, 2, 3, ..., k.$$
- $h_{i,bmu(x)}$ is the neighborhood function for a node i and $bmu(x)$ given by:
$$h_{i,bmu(x)} = e^{-\frac{d^2_{i,x}}{2\sigma^2}}$$

2.2.5 Online SOM algorithm

The online SOM algorithm performs the four steps of the training process above on a single input vector at a time. Thus it is a recursive and stepwise algorithm. It can be described as follows:

Algorithm 1: Online SOM

Input: X, k (total number of iterations)
Output: Trained $\{w_1, w_2, ..., w_m\}$

Initialize $w_1, w_2, ..., w_m$ randomly.
while $i < k$ **do**
  for x *in* X **do**
    find $bmu(x)$ by computing the Euclidean distances between x and $\{w_1, w_2, ..., w_m\}$.
    Determine the neighbors of $bmu(x)$.
    Update the weight vectors of $bmu(x)$ and its neighbors.
    $i \leftarrow i + 1$
  end
end

2.2.6 Batch SOM algorithm

Contrary to the online algorithm, the batch algorithm takes the entire dataset as one batch and update all weight vectors concurrently in one cycle. The batch algorithm is known to be faster and safer, and its convergence is considerably faster as well [11]. It also makes things simpler as it doesn't involve the learning rate parameter α as in the online algorithm. The weight vectors are updated as:

$$w_i(t+1) = \frac{\sum_{j=1}^{l} h_{i,bmu(x_j)}(t) x_j}{\sum_{j=1}^{l} h_{i,bmu(x_j)}(t)}, i \in \{1, 2, ..., m\} \tag{2.6}$$

where l is the total number of input vectors in X.

Algorithm 2: Batch SOM

Input: X, k (total number of epochs)
Output: Trained $\{w_1, w_2, ..., w_m\}$

Initialize $w_1, w_2, ..., w_m$ randomly.
while $i < k$ **do**
  find $bmu(x)$ by computing the Euclidean distances between x and $\{w_1, w_2, ..., w_m\}$ for all x.
  Determine the neighbors of $bmu(x)$ for all x.
  Update the weight vectors concurrently.
  $i \leftarrow i + 1$
end

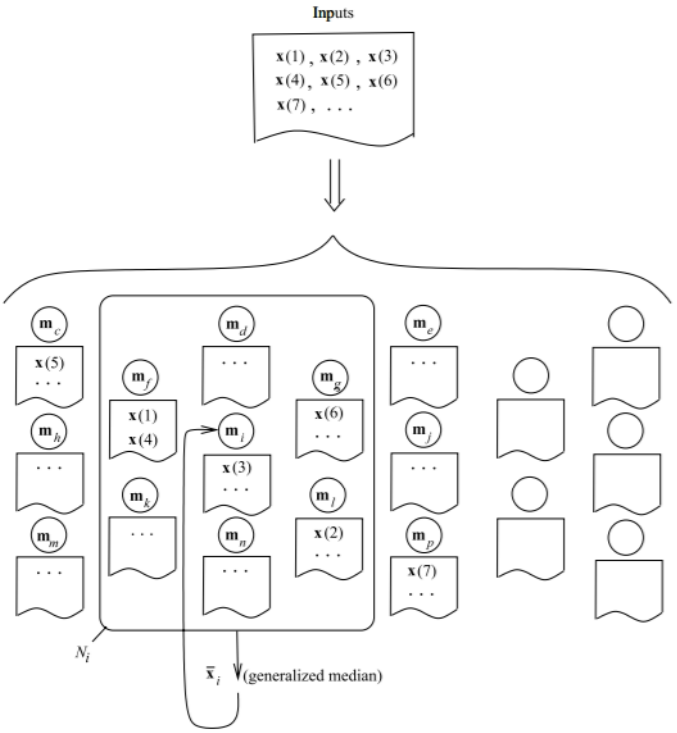


Figure 2.6: Illustration of one epoch in the batch SOM. Source: [11]

2.3 Evaluation methods

There are two widely used evaluation metrics for the SOM: quantization error and topographic error. The former measures the resolution of the SOM or the "goodness of its fit", whereas the latter assesses the topology preservation of the SOM. Quantization error is often used as a basic quality measure.

2.3.1 Quantization error

Quantization error is the average distance between each input vector and its BMU. In other words,

$$QE = \frac{1}{N}\sum_{i=1}^{N}\left\|x_i - w_{bmu(x_i)}\right\| \tag{2.7}$$

where N is the total number of input vectors in the dataset and $w_{bmu(x_i)}$ is the weight vector associated with the BMU of x_i. This metric measures the relationship between input vectors and their BMUs. Quantization error is a clustering metric, analogous to calculating the distance between each data point within a cluster and the centroid in k-means clustering.

2.3.2 Topographic error

One of the main properties of the SOM is topology preservation. Topographic error assesses the quality of topology preservation by finding the BMU and second-BMU of each input vector and comparing their positions as follows:

$$TE = \frac{1}{N}\sum_{i=1}^{N} t(x_i) \tag{2.8}$$

$$t(x_i) = \begin{cases} 0 & \text{if } bmu(x) \text{ and } second\_bmu(x) \text{ are adjacent} \\ 1 & \text{otherwise} \end{cases}$$

In other words, topographic error measures the proportion of input vectors whose BMU and second-BMU are not adjacent.

2.3.3 U-matrix

U-matrix is a visual representation of the SOM which displays the distances between adjacent SOM nodes in a grayscale image. A lighter coloring between nodes indicates the weight vectors associated with the nodes are close to each other, and vice versa. U-matrix is a useful tool that provides a visual topology representation of the SOM.

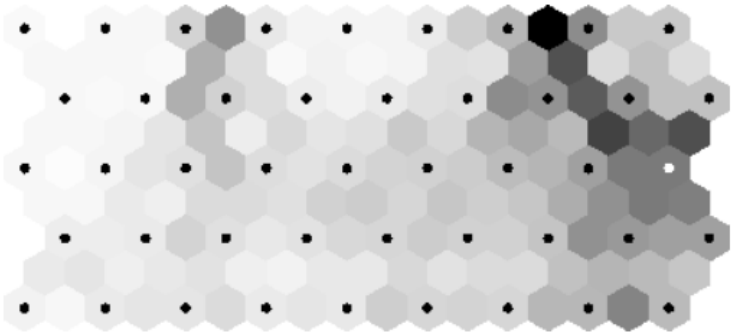


Figure 2.7: Example of a U-matrix. Source: [13]

Chapter 3

Experiments

3.1 Aim

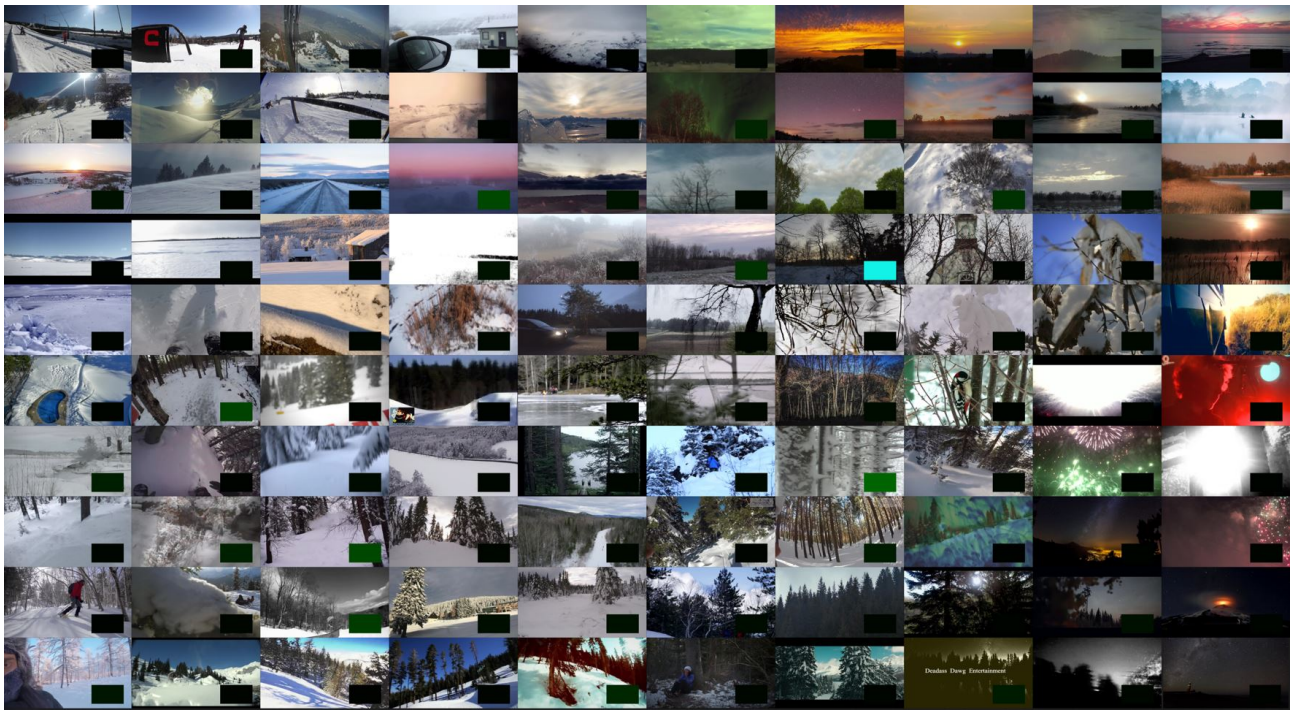


Figure 3.1: Output of Plain SOM. The target frame of the query is emphasized with a mint rectangular patch at its bottom right corner. The other frames have similar rectangular patches, whose colors are represented as the G value of an RGB triplet (with R and B values being zero).

Although the SOM is a powerful clustering and dimensionality reduction algorithm, it poses a notable problem for our use case: known-item search in video frames. Our video frames have relevance scores assigned with respect to a target image and we wish to display top-scoring images in the

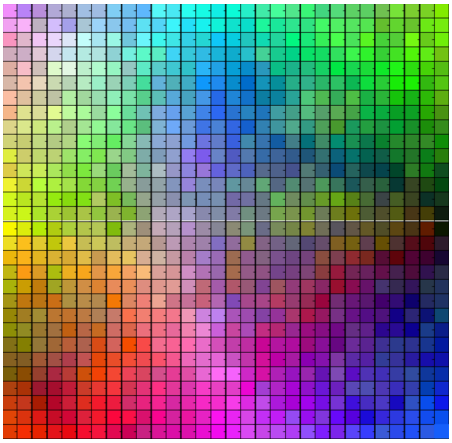


Figure 3.2: Output of 30-by-30 Plain SOM for randomly generated colors

upper rows of output display, and vice versa. The SOM only takes into account the similarity of given data; it organizes the input items solely based on the similarity of input items in the original feature space, whether or not there exists some ranking among them. This is demonstrated in Figure 3.1, in which higher-scoring frames have patches in lighter shades of green (except for the target frame, which has the mint patch). In the language of IR systems, the original SOM only has the "exploration" capabilities as opposed to exploitation. The display utilizing only the exploitation capabilities is often called Top-k display, in which items are selected and displayed solely in the order of their relevance.

Our goal is to modify the SOM so that it considers the individual relevance scores as well as local similarity in the original feature space (i.e. our goal is find a middle ground between the original SOM and Top-k display). With this in mind, we devised two variants of the SOM: *Rating dimension SOM* and *Rating-weighted SOM*. For the sake of clarity, the original SOM will be denoted as *Plain SOM* henceforth.

For illustration purposes, we will be training our SOM variants with randomly generated colors as shown in Figure 3.2. This time, we train a 10-by-10 SOM on 1000 randomly generated colors represented by RGB vectors (with corresponding RGB values following a uniform distribution in the range [0, 1]). To introduce the notion of rating to this data, we select a target color, and assign a relevance score to each color image based on its similarity to the target color. This leaves us with a realistic use case that is much simpler and more intuitive.

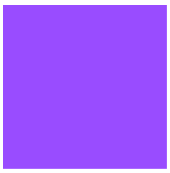


Figure 3.3: An example target color for our task, represented by RGB triplet (0.6, 0.3, 1.0)

Probabilistic sample selection

While training the SOM on the entire dataset improves the diversity of the display, it often hinders more relevant images from being displayed. This does not benefit our use case in which the user intends to exploit results relevant to their query. To tackle this problem, Kratochvil et. al devised a probabilistic sample selection procedure, in which each item from the dataset is selected with a probability proportional to its respective relevance score [2, 4]. An alternative approach to this may be to simply select top N images. However, SOM with porbabilistic sample selection may produce more diverse displays, especially if N is small. In either case, pre-processing items from the original dataset would result in displays with more relevant objects.

3.2 Rating dimension SOM

The main idea behind Rating dimension SOM (RDSOM) is adding a new dimension of relevance scores to both the training data and the weight vectors of the SOM, and keeping the orientation of the SOM close to that of the data with respect to the new dimension. For each input vector x from our dataset, we add a corresponding relevance score as its new dimension. Similarly, for each weight vector of the SOM, we add a relevance score as its new dimension based on its position; since we want top-scoring images to be displayed in top rows, we assign the highest query score to the top left node, and so forth. With input images and their respective scores, we have the desired final layout of scores in the SOM. With this in mind, we reorganize the SOM to recognize and respect that layout during training.

There are a few approaches to this we can take, which we will explore in this section.

3.2.1 Initial bias

Prior to training, adding initial bias to the weight vectors to match the orientation of top-k ordering of images can help the SOM arrange the images in the order of importance as well as find the topology of given data. We assign the new dimension of each weight vector the relevance score of the image that would appear at the corresponding node position in top-k ordering.

As for the new dimension of our data, we need to ensure that its variance is significant enough for the SOM. Otherwise, the SOM will disregard the score distinction. Hence, we scale up the rating dimension so that its variance is greater than the rest of the features combined by dividing it with its standard deviation, which adjusts the variance to 1.

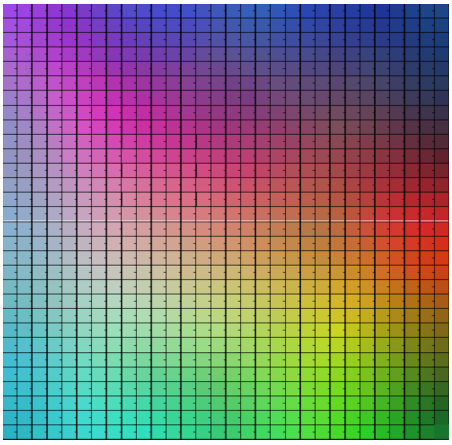


Figure 3.4: Trained weight vectors of the SOM with initial bias

3.2.2 Re-biasing

It may happen that we need to bias the SOM during the training process. For example, with the initial bias method above, the SOM may unexpectedly move the top-scoring images to lower positions because we are not adding any bias to the SOM during the training process. For this reason, we propose an alternative approach we call "re-biasing", which biases the rating dimension of the weight vectors during training. The core idea of re-biasing is nearly identical to that of initial bias. However, since this method is applied during training, it poses a few questions to be answered: how far through training do we halt re-biasing (i.e. when is the optimal timing to stop re-biasing)? In what manner do we apply re-biasing? For example, do we apply it to all of the weight vectors as re-biasing, or perhaps only to ones in the top and bottom rows?

As for the optimal timing to halt re-biasing, it is important that we let the SOM train on its own (without bias) towards the end of training. As mentioned in Chapter 2, σ determines the radius of neighborhood (informally, the "area of influence") and its value decays over time during training. The last iterations with low σ allow the SOM to spread out properly and find a better fit for the data. For that reason, re-biasing near the end of the training process can hinder such a process. Thus we experimented with the following thresholds (which we call cut-points throughout the rest of this paper): 0.1 (i.e. re-biasing up to 10% of the total number of iterations), 0.25, 0.5, 0.75 and 0.9.

Nonetheless, several options remain for the very method of introducing bias to the weight vectors during training. We experimented with adding bias from the top m images of our dataset (e.g. in case of a 10-by-10 SOM, use top 100 images to affect the score dimension of the weight vectors), and adding bias to the top row and bottom row only (e.g. use top 10 images to bias the weight vectors in the top row of SOM, and bottom 10 images to bias the bottom row). Of these two sub-variants of Rating Dimension SOM, the former will be denoted in results as *RDSOM-all* (short for all rows; also denoted in subsequent figures as *full_re-biasing*), while the latter will be denoted as *RDSOM-first_last* (short for first and last rows only).

Algorithm 3: RDSOM-all

Input: X (dataset), s (scores), k (total number of iterations), c (cut-point)
Output: Trained $\{w_1, w_2, ..., w_m\}$

Initialize $w_1, w_2, ..., w_m$ randomly, with n+1 dimensions.
Add s column to X to make n+1 dimensions.
Sort X based on the (n+1)-th dimension.
while $i < k$ **do**
  if $i < k * c$ **then**
    $r \leftarrow \sqrt{1 - \frac{i}{k*c}}$
    $w_j[n+1] \leftarrow r * x_j[n+1] + (1-r) * w_j[n+1] \; \forall j \in 1, 2, ..m$
  end
  for x *in* X **do**
    find $bmu(x)$ by computing the Euclidean distances between x and $\{w_1, w_2, ..., w_m\}$.
    Determine the neighbors of $bmu(x)$.
    Update the weight vectors of $bmu(x)$ and its neighbors.
    $i \leftarrow i + 1$
  end
end

Once training is complete, the BMU of each input vector is determined in order to display the output. It is possible that multiple items are be mapped to the same node, leaving some nodes empty. In such a case, we select the next closest node possible. We repeat this process until we find an empty node to assign the item to.

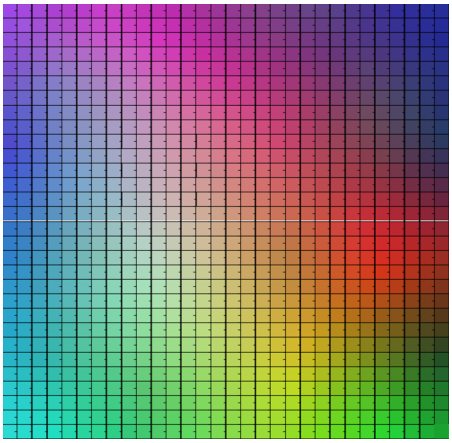


Figure 3.5: Trained weight vectors with re-biasing applied with *cut-point* $= 0.75$ (i.e. up to the third quarter of training). High-scoring, purple-shaded images have been moved noticeably further up via re-biasing.

3.3 Rating-weighted SOM

Rating-weighted SOM (RWSOM) revolves around modifying the distance metric to take the rating into consideration. The Euclidean distance is the default metric for the SOM, which we calculate for each pair of feature vectors ("images") and weight vectors ("nodes"). At each iteration, we select the node with minimal distance from the current image. Let us denote the Euclidean distance between a feature vector and a weight vector as $L2(node, image)$. The main idea behind Rating-weighted SOM is creating a new metric which adds bias to $L2(node, image)$:

$$dist(node, image) = L2(node, image) + f(node, image) \tag{3.1}$$

where f is some function that adds bias depending on the rating (or ranking) of the corresponding node and image. Our initial question was whether to use ordinal ranks or original ratings (relevance scores) for Rating-weighted SOM. We opted for ratings, as our original dataset contains 20,000 images and the output of f can grow very large, overwhelming any local similarity. To determine potential choices for f, we need to consider its desirable characteristics:

- f must recognize the relative importance of nodes (e.g. the top left node is very important, the rightmost node in the third row isn't as important, and the bottom right node is completely unimportant).
- if both the image and the node have high ratings, f must "reward" the node by setting the distance low, making it more likely for the BMU selection process to select the associated node.

On that note, we devised node ratings and node weights. The notion of node ratings was added in order to specify the relative importance of nodes at different positions. The top left node will have the highest rating of 1, and the bottom right node will have the lowest rating of 0. For all nodes in between, we generated a set of evenly spaced numbers within interval (0, 1), and assigned the values in decreasing order starting with nodes in the top row. Furthermore, we assigned a node weight $\frac{1}{node\_rating_i}$ for the i-th node (i = 1, 2,...,m), where the first node is the top left node, the second is the node next to it, and the m-th node is the bottom right node.

Putting these notions together, we devised the following functions for f:

1. $f(node_i, image_j) = node\_weight_i * \|node\_rating_i - image\_rating_j\|$
2. $f(node_i, image_j) = node\_weight_i * \dfrac{|node\_rating_i - image\_rating_j|}{max(node\_rating_i, image\_rating_j)}$
3. $f(node_i, image_j) = node\_weight_i * \dfrac{|node\_rating_i - image\_rating_j|}{min(node\_rating_i, image\_rating_j)}$
4. $f(node_i, image_j) = node\_weight_i * \log(|node\_rating_i - image\_rating_j|)$

In results, Rating-weighted SOM with the above choices of f will be denoted as *RWSOM-euc* (short for Euclidean distance), *RWSOM-frac_min* (short for fractional difference with minimum), *RWSOM-frac_max* (short for fractional difference with maximum), *RWSOM-log* (short for logarithm, also denoted in subsequent figures as log_abs), respectively.

In Chapter 2, the BMU for x was selected as follows in Plain SOM:

$$bmu(x) = \underset{i}{\operatorname{argmin}} \|x - w_i\|, i = 1, 2, ..., m \tag{3.2}$$

Combining this with f, we have the following formula for BMU selection:

$$bmu(x) = \underset{i}{\operatorname{argmin}}(\beta * L2(x, w_i) + (1 - \beta) * f) \tag{3.3}$$

where β is a hyperparameter which helps us tune the trade-off between $Plain\ SOM$ and Top-K display [4].

3.4 Evaluation protocol

Evaluating the SOM's display is quite a complicated task. Since all of the images are intended for humans, it is important that the output is visibly suitable for the given query - which is very difficult to quantify and can vary from person to person. Additionally, simply by looking at the images, it's not often clear how organized the results are with respect to the relevance and similarity of the data. Consequently, any claims of the "optimal" evaluation metric would be highly disputable. Nevertheless, we can put together several practical measures in an effort to evaluate the goodness of our output.

Due to the nature of our task, we need to evaluate how well our SOM variants preserved both the relevance and the local similarity of the data. Since our goal is nearly identical to that of Peška and Lokoč [4], note that this section of the paper was heavily inspired by their evaluation protocol: relevance scores of candidate items were computed as $r_i = e^{-exp*L2(x_t+\mathbf{b},x_i)}$, where *t* is an arbitrary target image and *exp* is a scoring hyperparameter tuning the steepness of the relevance model, and **b** is a random noise vector selected from a normal distribution with $\mu = 0$ and $\sigma = 0.1$ in case of RGB colors, and $\mu = 0$ and $\sigma = 0.2$ for the video frames [4]. Similarly, we experimented with $\beta \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.2, 0.5, 0.8, 0.9\}$ for Rating-weighted SOM, *cut-point* $\in \{0.1, 0.25, 0.5, 0.75.0.9\}$ for Rating dimension SOM, $exp \in \{5, 10\}$ for the color dataset and $exp = 10$ for the frame dataset.

Outputs of the following SOM variants were evaluated against Top-K display:

- Plain SOM
    1. *Plain SOM*
    2. *Plain SOM+prob*

- Rating dimension SOM
    1. Rating dimension SOM with initial bias only
        (a) *RDSOM-initial_bias*
        (b) *RDSOM-initial_bias+prob*
    2. Rating dimension SOM with re-biasing
        2.1. Rating dimension SOM with full re-biasing
            (a) *RDSOM-all* (followed by *cut-point* denoted as a suffix; e.g. *RDSOM-all_0.1*)
            (b) *RDSOM-all+prob* (e.g. *RDSOM-all_0.1+prob*)
        2.2. Rating dimension SOM with re-biasing on first and last rows only
            (a) *RDSOM-first_last* (e.g. *RDSOM-first_last_0.1*)
            (b) *RDSOM-first_last+prob* (e.g. *RDSOM-first_last_0.1+prob*)
- Rating-weighted SOM
    1. *RWSOM* (followed by a choice of f and β; e.g. *RWSOM-log_0.001*)
    2. *RWSOM+prob* (e.g. *RWSOM-log_0.001_+prob*)

where *RDSOM* denotes Rating dimension SOM, *RWSOM* denotes Rating-weighted SOM, and *+prob* denotes probabilistic sample selection. Variants without the *+* symbol run on *all* images from our dataset (i.e. without any image selection process prior to training).

Peška and Lokoč formulated three requirements to determine good SOM output: relevance, diversity, and topology preservation [4]. The requirements state that the output must contain the most relevant results in the most prominent areas of the display, and that it must preserve the diversity and topology of given data. On that note, they put together three evaluation metrics: nDCG, *div(all)*, and *divRatio*.

NDCG evaluates the display in terms of relevance preservation, whose values closer to 1 indicate that the SOM display resembles Top-K display.

Both *div(all)* and *divRatio* are obtained by comparing the distance of images on the output screen and their distance in the feature space. *div(all)* evaluates overall diversity (coverage) of the display, which is calculated as mean Euclidean distance with respect to all pairs of displayed items. Note that higher *div(all)* values indicate more diverse results.

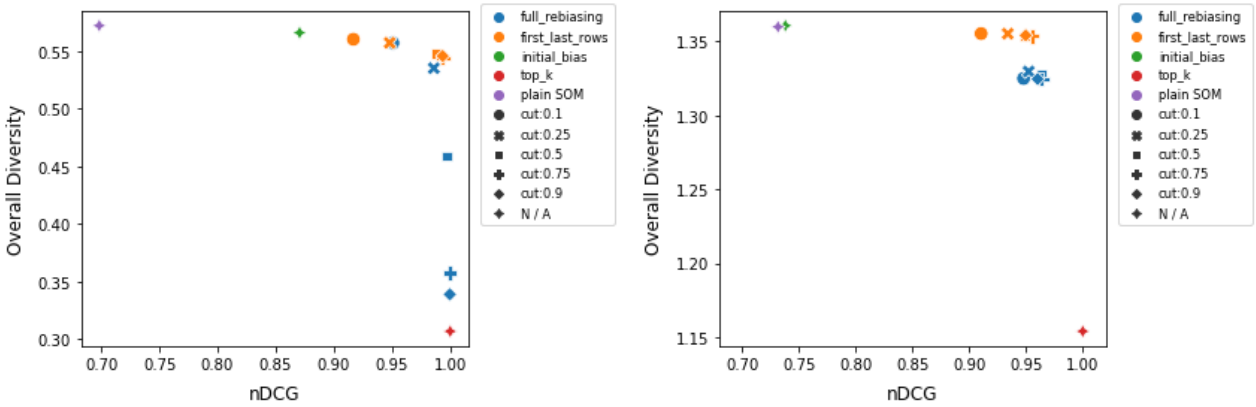
Finally, *divRatio* denotes the ratio between neighbors' diversity and overall diversity (i.e. $divRatio = div(neighbors)/div(all)$), and measures orderliness of the display. For each node on the SOM display, all nodes within its "immediate surroundings" are considered its neighbors. For example, the node at index [0, 0] has three neighbors: at [0, 1], [1, 0], and [1, 1]. *div(neighbors)* is then calculated as mean Euclidean distance between nodes and their neighbors:

$$divRatio = \sum_{\forall i \in [m], j \in A_i} \frac{L2(w_i, w_j)}{|A|} \tag{3.4}$$

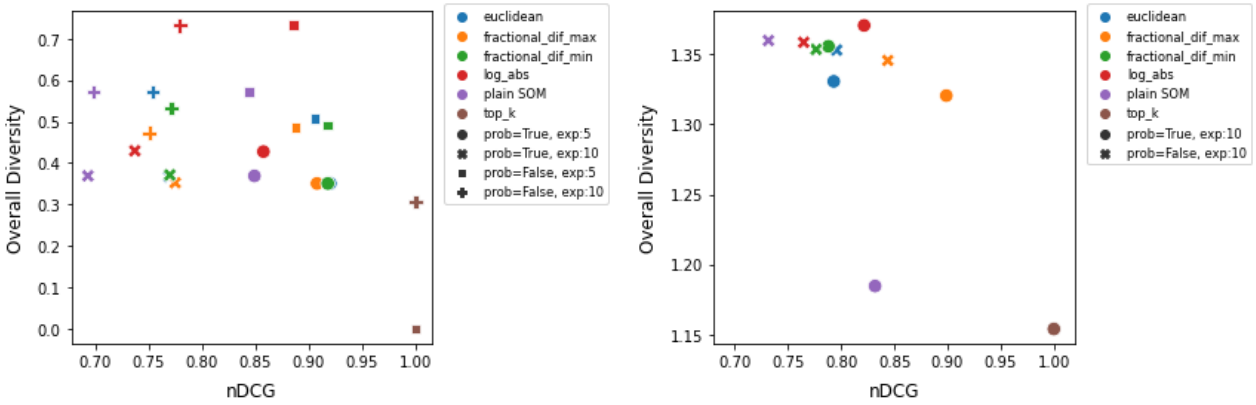
where A_i is the set of all adjacent nodes for the i-th node. Note that lower *divRatio* values indicate more orderly results [4].

Chapter 4

Results and Discussion



(a) Rating dimension SOM with $exp = 10$ and *cut-point* represented by markers



(b) Rating-weighted SOM

Figure 4.1: Results of Rating dimension SOM and Rating-weighted SOM trained on RGB colors (left) and video frames (right). X-axis depicts nDCG (i.e. exploitation capability) while Y-axis depicts the overall diversity of selected images (i.e. exploration capability) [4], and the colors of the data points depict the corresponding SOM (sub-)variants.

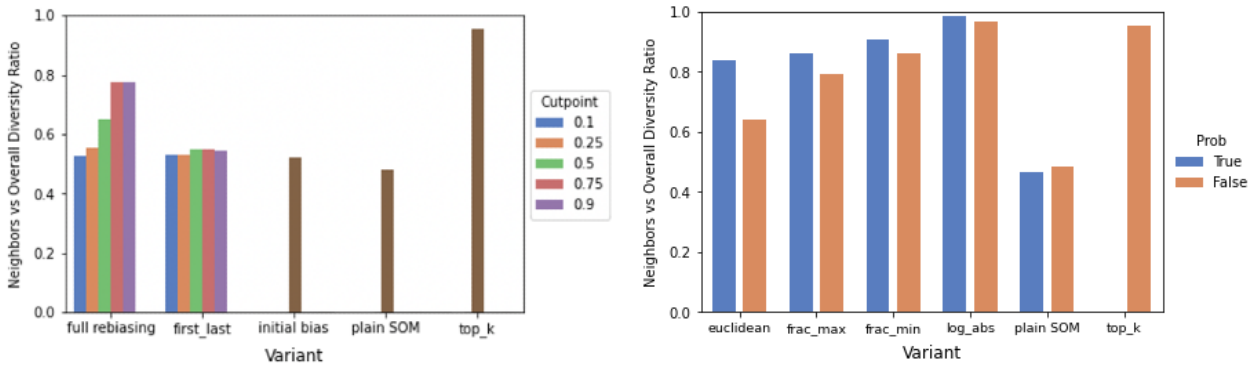


Figure 4.2: *divRatio* measured on Rating Dimension SOM and Rating-weighted SOM with $exp = 10$. *cut-point* was applied where applicable.

In this chapter, we will evaluate our SOM variants using the evaluation metrics described in Chapter 3. Subsequently, we will examine example displays of the SOM variants trained on RGB colors to visually compare the results. We will then compare Rating dimension SOM and Rating-weighted SOM individually with different hyperparameter settings.

Figure 4.1 and Figure 4.2 show overall results for the RGB color dataset and video frame dataset, measured by our evaluation metrics. Note that since both Rating dimension SOM and Rating-weighted SOM have quite a few sub-variants and combinations of hyperparameters, we restricted the experiment settings to $exp = 10$ and without probabilistic selection in case of Rating dimension SOM. As for Rating-weighted SOM, its sub-variants were evaluated by the mean values obtained from all possible β values.

In both figures, we can observe that Rating dimension SOM and Rating-weighted SOM bridge the gap between Plain SOM and Top-k display. This aligns with Rating-aware SOM [4]. However, Figure 4.1 (b) indicates that probabilistic sample selection didn't present sufficient help in improving nDCG with respect to the rating-weighted distance metrics. As for Rating dimension SOM, increasing *cut-point* values generally resulted in improved nDCG, although it caused a sharp decrease in *div(all)* for the color data. Furthermore, it's worth noting that *RDSOM-first_last* showed a clear improvement on nDCG while preserving the overall diversity of the data fairly well.

Let us explore our SOM variants in mote detail by simulating a known-item search scenario with a target image and relevance scores (ratings) assigned to the color dataset. As shown in Figure 4.3, it's clear that Rating-weighted SOM's output is quite disorderly (especially *RWSOM-log*), in contrast to Rating dimension SOM. However, while the upper half of its display is cluttered, it consistently displayed a fairly high-scoring image in the top left node in the current experiment setting. As for Rating dimension SOM, *RDSOM-initial_bias* displayed high-scoring images in upper rows while preserving the diversity of the data. However, top-scoring images are not displayed in the top left corner, which illustrates why initial bias alone is often not sufficient. Unsurprisingly, among RDSOM variants, *RDSOM-all* was most successful in displaying high-scoring images.

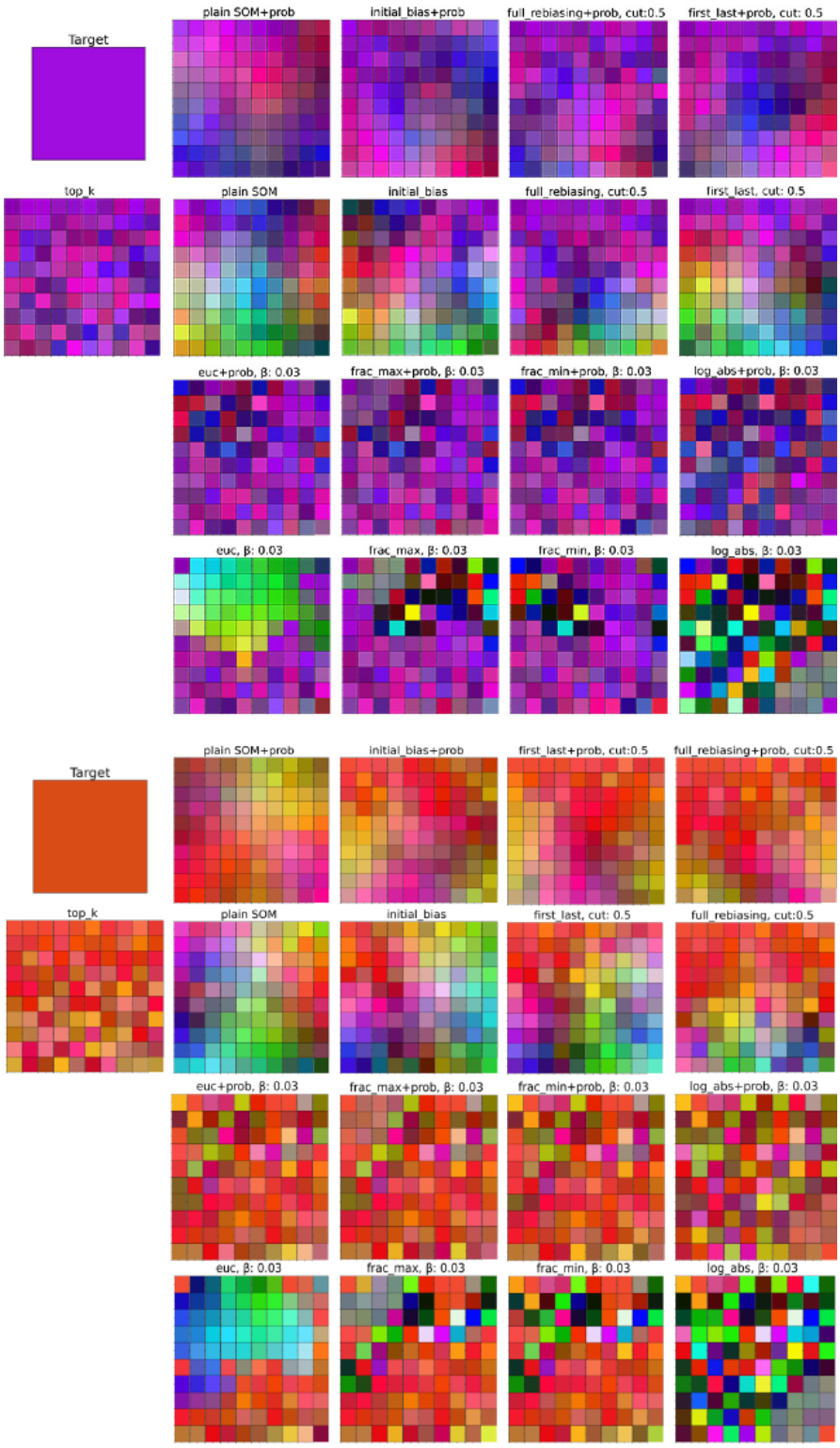


Figure 4.3: Example display of our SOM variants trained on RGB colors. All methods utilized $\beta = 0.03$, $exp = 10$, $cut\text{-}point = 0.5$ where applicable. Starting from the top row, Plain SOM, RDSOM-initial_bias, RDSOM-all, RDSOM-first_last, RWSOM-euc, RWSOM-frac_max, RWSOM-frac_min, and RWSOM-log are displayed in order.

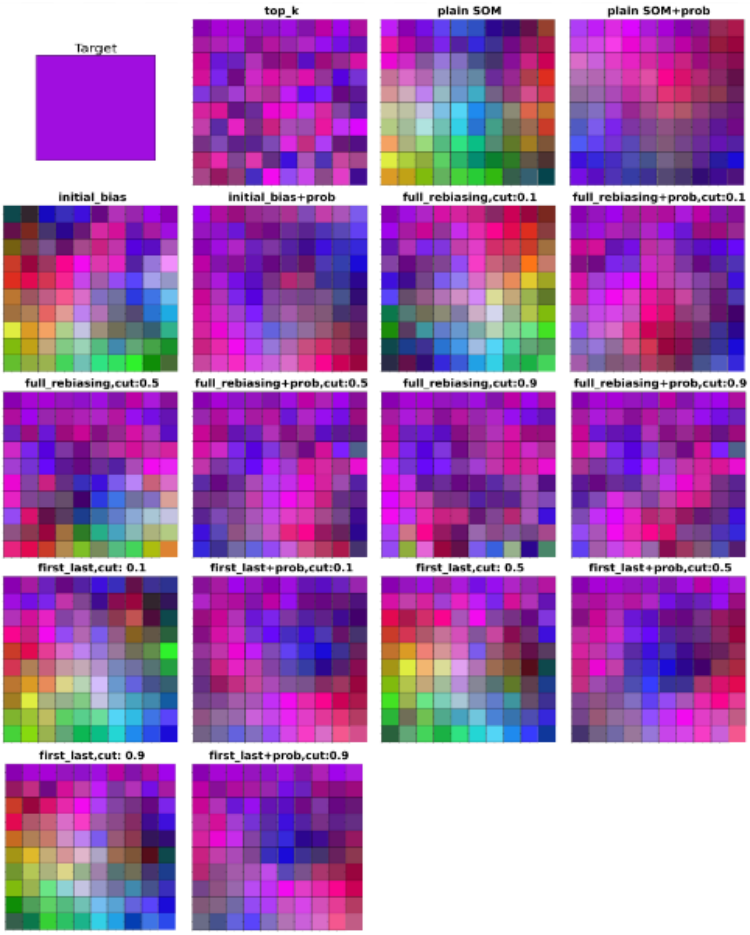


Figure 4.4: Example display of Rating dimension SOM with probabilistic sample selection and various *cut-point* values. *RDSOM-all* and *RDSOM-first_last* are displayed in the order of RDSOM_0.1, RDSOM_0.1+prob, RDSOM_0.5, RDSOM_0.5+prob, RDSOM_0.9, RDSOM_0.9+prob. Note that Figure 4.4 and Figure 4.5 are a continuation of the first example in Figure 4.3.

Figure 4.4 depicts the display of Rating dimension SOM with probabilistic sample selection and different *cut-point* values. Without probabilistic selection, it is apparent that *RDSOM-all* covers the entire screen with highly relevant images with higher *cut-point* while *RDSOM-first_last* tends to populate only the upper part of the display with such images.

Let us now consider Rating-weighted SOM in more detail. Since *RWSOM-log* failed to provide marginally satisfactory results, it will be omitted from further experiments.

As previously mentioned, Figure 4.5 indicates that Rating-weighted SOM usually displays a high-scoring image in the top left node, despite (possibly) dissimilar images adjacent to it. We suspect that this is potentially caused by low σ, currently fixed to 1.0. Furthermore, generally speaking, higher β appears to provide more orderly and smooth displays while moderately preserving the relevance of given data. This is in line with Rating-aware SOM as evaluated by Peška and Lokoč [4].

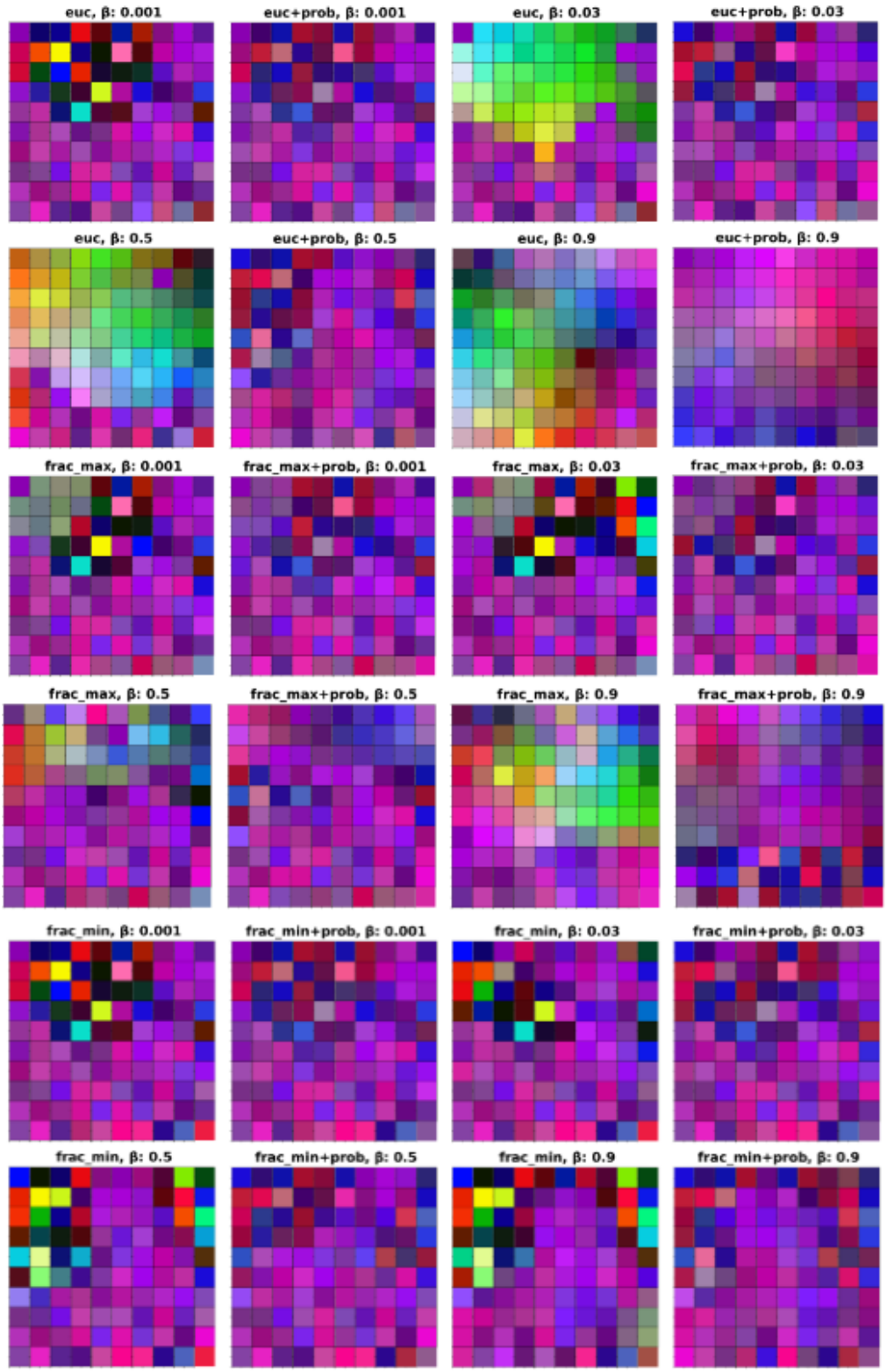


Figure 4.5: Example display of Rating-weighted SOM with probabilistic sample selection and various β values

Discussion

Although Rating-weighted SOM showed a noticeable improvement with prior-to-training probabilistic sample selection, our study has some potential limitations.

Firstly, since there exist many sub-variants of our variants and thus many combinations of hyperparameter values, α (learning rate) and σ (neighborhood radius) of our SOM variants were fixed to 1.0. As briefly mentioned above, it is possible that Rating-weighted SOM can improve with different hyperparameter settings e.g. custom distance metrics with higher σ. Tuning β may help us find a suitable middle ground between Plain SOM and Top-k display, as demonstrated in Figure 4.6.

Secondly, as stated in the Introduction, this study considered only the row-first ordering of nodes; whereas Peška and Lokoč considered both the row-first ordering and triangular ordering when performing their experiments [4]. The "golden triangle" pattern is known to closely resemble the user's eye movement when viewing images online [14, 15]. For our use case, the triangular ordering better serve as the "prominency ordering" of nodes.

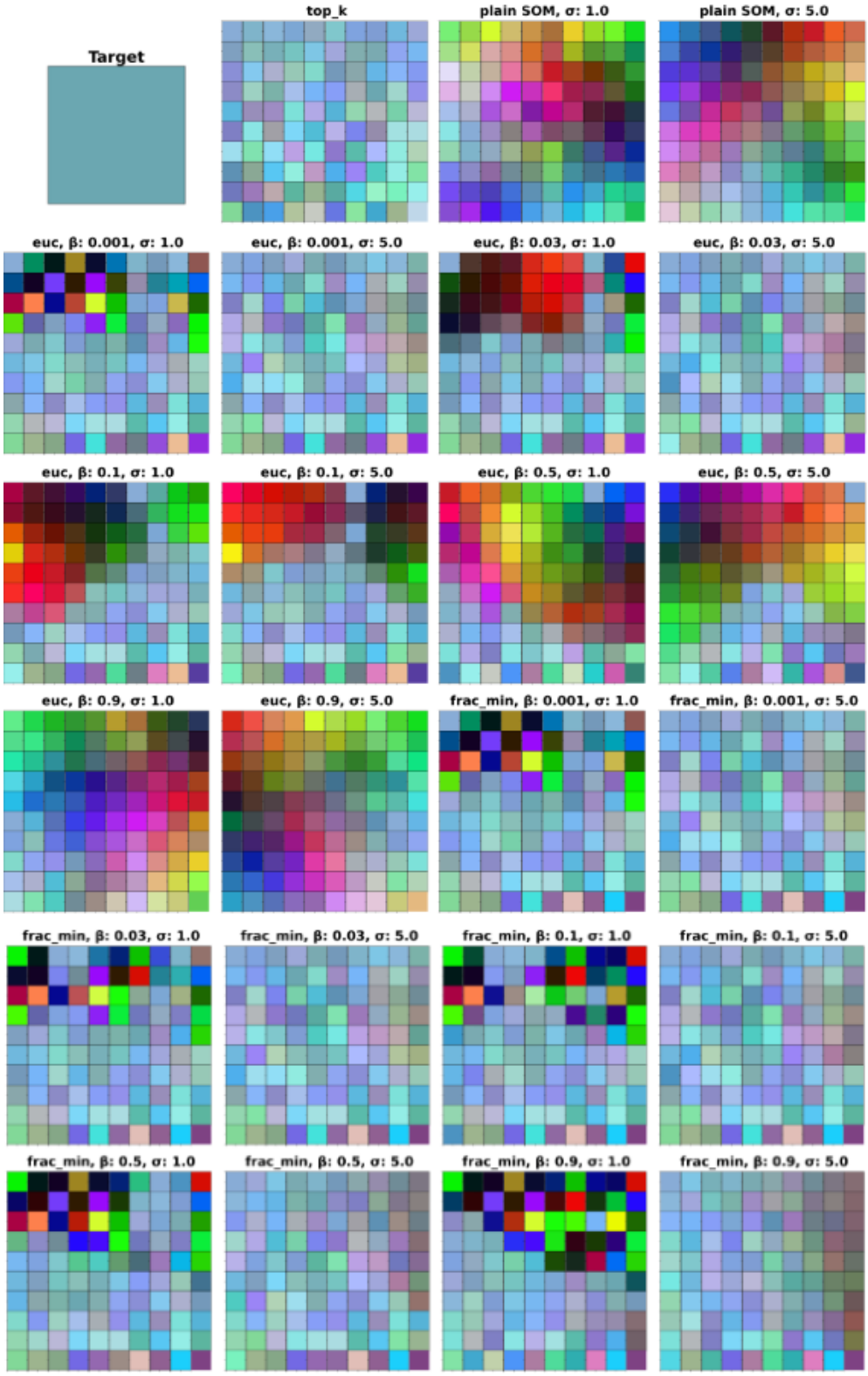


Figure 4.6: Example display of Rating-weighted SOM with different combinations of β and σ values. For the sake of brevity, probabilistic sample selection was omitted.

Conclusion

In this work, we proposed two rating-aware variants of SOM and their sub-variants in order to incorporate the relevance of given data in SOM: Rating dimension SOM and Rating-weighted SOM. We used three evaluation metrics proposed by Peška and Lokoč [4] to evaluate our results, aimed at measuring the diversity, relevance and the degree of topology preservation of the display. Furthermore, we examined example displays with color images (simulating a known-item search scenario) to visually compare the results.

Our results showed an improvement over Plain SOM with respect to these metrics, although they also indicated there's room for further improvement.

Future work

As mentioned in Chapter 4, different hyperparameter settings may help improve the results; particularly for Rating-weighted SOM. Furthermore, we may be able to find a better choice of f that better takes the relevance of given data into account while preserving the local similarity.

As for the prominence ordering of nodes mentioned in Chapter 4, we may also modify our SOM variants such that it considers the triangular ordering (e.g. as for Rating dimension SOM, by re-biasing the weight vectors in a different order). Furthermore, in a similar manner as β, Rating dimension SOM may also be tuned by some hyperparameter.

Bibliography

[1] Maryam Karimzadehgan and ChengXiang Zhai. "Exploration-Exploitation Tradeoff in Interactive Relevance Feedback". In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. CIKM '10. Toronto, ON, Canada: Association for Computing Machinery, 2010, 1397–1400. ISBN: 9781450300995. DOI: 10.1145/1871437.1871631. URL: https://doi.org/10.1145/1871437.1871631.

[2] Miroslav Kratochvil et al. "SOMHunter: Lightweight Video Search System with SOM-Guided Relevance Feedback". In: *Proceedings of the 28th ACM International Conference on Multimedia*. New York, NY, USA: Association for Computing Machinery, 2020, 4481–4484. ISBN: 9781450379885. URL: https://doi.org/10.1145/3394171.3414542.

[3] Hansenclever F. Bassani and Aluizio F. R. Araújo. "Dimension Selective Self-Organizing Maps for clustering high dimensional data". In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252416.

[4] Ladislav Peška and Jakub Lokoč. "Rating-Aware Self-Organizing Maps". In: *MultiMedia Modeling*. Ed. by Björn Þór Jónsson et al. Cham: Springer International Publishing, 2022, pp. 119–130. ISBN: 978-3-030-98358-1.

[5] Luca Rossetto et al. "V3C–a research video collection". In: *International Conference on Multimedia Modeling*. Springer. 2019, pp. 349–360.

[6] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: https://books.google.cz/books?id=EoYBngEACAAJ.

[7] G. James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2014. ISBN: 9781461471370. URL: https://books.google.cz/books?id=at1bmAEACAAJ.

[8] T. Kohonen, T.S. Huang, and M.R. Schroeder. *Self-Organizing Maps*. Physics and astronomy online library. Springer Berlin Heidelberg, 2001. ISBN: 9783540679219. URL: https://books.google.cz/books?id=e4igHzyfO78C.

[9] Christoph von der Malsburg. "Self-organization of orientation sensitive cells in the striate cortex". In: *Biological Cybernetics* 14 (Jan. 1973), pp. 85–100. DOI: 10.1007/BF00288907.

[10] Kosala Sananthana. *How to implement Kohonen's Self Organizing Maps*. https://towardsdatascience.com/how-to-implement-kohonens-self-organizing-maps-989c4da05f19. 2020.

[11] Teuvo Kohonen. “Essentials of the self-organizing map”. In: *Neural Networks* 37 (2013). Twenty-fifth Anniversay Commemorative Issue, pp. 52–65. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.09.018. URL: https://www.sciencedirect.com/science/article/pii/S0893608012002596.

[12] Ayodeji A. Akinduko, Evgeny M. Mirkes, and Alexander N. Gorban. “SOM: Stochastic initialization versus principal components”. In: *Information Sciences* 364-365 (2016), pp. 213–221. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2015.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0020025515007318.

[13] Jaakko Hollmén. *Applications of learning and intelligent systems*. 1996. URL: https://users.ics.aalto.fi/jhollmen/dippa/node24.html.

[14] Laura Granka, Matthew Feusner, and Lori Lorigo. *Eyetracking in Online Search*. 2008.

[15] R.I. Hammoud. *Passive Eye Monitoring: Algorithms, Applications and Experiments*. Signals and Communication Technology. Springer Berlin Heidelberg, 2008. ISBN: 9783540754121. URL: https://books.google.co.kr/books?id=k0VGgU0lHroC.

List of Figures

1.1 Example of nDCG calculation . . . 7
1.2 Example of linear regression. Source: [7] . . . 10
1.3 Example of k-means clustering with different k values. The crosses represent the centroids of each cluster. . . . 11
1.4 A very simple ANN. f denotes some function which processes each input node with its weight and produces an output y. . . . 11

2.1 Structure of SOM. x denotes an input vector with n dimensions, and the grid on the right depicts the SOM. The lines in the middle depict weight vectors associated with each SOM node, connected to each dimension of x. . . . 13
2.2 Example usage of SOM demonstrating SOM's topology preservation, in clockwise direction from top left. Source: [10] . . . 13
2.3 Illustration of neighborhood radius decay over time during training. As the radius decreases, fewer nodes are influenced by the change caused by the BMU. . . . 15
2.4 Example usage of SOM . . . 16
2.5 BMU (the node in the center of the SOM) and nodes within its neighborhood radius 17
2.6 Illustration of one epoch in the batch SOM. Source: [11] . . . 19
2.7 Example of a U-matrix. Source: [13] . . . 21

3.1 Output of Plain SOM. The target frame of the query is emphasized with a mint rectangular patch at its bottom right corner. The other frames have similar rectangular patches, whose colors are represented as the G value of an RGB triplet (with R and B values being zero). . . . 22
3.2 Output of 30-by-30 Plain SOM for randomly generated colors . . . 23
3.3 An example target color for our task, represented by RGB triplet (0.6, 0.3, 1.0) . . . 23
3.4 Trained weight vectors of the SOM with initial bias . . . 25
3.5 Trained weight vectors with re-biasing applied with *cut-point* $= 0.75$ (i.e. up to the third quarter of training). High-scoring, purple-shaded images have been moved noticeably further up via re-biasing. . . . 26

4.1 Results of Rating dimension SOM and Rating-weighted SOM trained on RGB colors (left) and video frames (right). X-axis depicts nDCG (i.e. exploitation capability) while Y-axis depicts the overall diversity of selected images (i.e. exploration capability) [4], and the colors of the data points depict the corresponding SOM (sub-)variants. . . . 32
4.2 *divRatio* measured on Rating Dimension SOM and Rating-weighted SOM with $exp = 10$. *cut-point* was applied where applicable. 33
4.3 Example display of our SOM variants trained on RGB colors. All methods utilized $\beta = 0.03$, $exp = 10$, $cut\text{-}point = 0.5$ where applicable. Starting from the top row, Plain SOM, RDSOM-initial_bias, RDSOM-all, RDSOM-first_last, RWSOM-euc, RWSOM-frac_max, RWSOM-frac_min, and RWSOM-log are displayed in order. . . 34
4.4 Example display of Rating dimension SOM with probabilistic sample selection and various *cut-point* values. *RDSOM-all* and *RDSOM-first_last* are displayed in the order of RDSOM_0.1, RDSOM_0.1+prob, RDSOM_0.5, RDSOM_0.5+prob, RDSOM_0.9, RDSOM_0.9+prob. Note that Figure 4.4 and Figure 4.5 are a continuation of the first example in Figure 4.3. 35
4.5 Example display of Rating-weighted SOM with probabilistic sample selection and various β values 36
4.6 Example display of Rating-weighted SOM with different combinations of β and σ values. For the sake of brevity, probabilistic sample selection was omitted. 38

List of Acronyms

ANN Artificial Neural Network

BMU Best Matching Unit

IR Information Retrieval

KIS Known-item Search

NDCG Normalized Discounted Cumulative Gain

SOM Self-organizing Map

RDSOM Rating Dimension Self-organizing Map

RWSOM Rating-weighted Self-organizing Map