Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Prevention of occurrence of dead units in self-organizing maps

Master thesis

2019
Bc. Jakub Novák

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Prevention of occurrence of dead units in self-organizing maps

Master thesis

Study program:	Informatics
Field of study:	2508 Informatics
Department:	Department of Informatics
Supervisor:	doc. RNDr. Martin Takáč, PhD.

Bratislava, 2019
Bc. Jakub Novák

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Jakub Novák
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Prevention of occurrence of dead units in self-organizing maps

Annotation: One of the problems of self-organizing maps (SOM) is occurrence of so-called "dead units" that effectively lower the capacity of the SOM. Usually due to random weight initialization, some neurons have weights far from any data, hence never win the competition and rarely get any training that would pull them out of bad regions, hence they become "dead". The goal of this thesis is to review existing approaches to the problem, suggest several SOM modifications, implement simulations and analyse to what extent the proposed modifications succeed in eliminating dead units. This also includes exploring the space of free parameters of these modifications and finding the best values.

Literature: M. Van Hulle (2000): Self-organizing Maps. In: Handbook of Natural Computing, pp 585-622.
T. Villmann et al (2014): Advances in Self-Organizing Maps and Learning Vector Quantization. Proceedings of the 10th International Workshop WSOM.

Supervisor:	doc. RNDr. Martin Takáč, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	14.12.2016

Approved: 19.12.2016 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgement: First and foremost, I would like to thank my supervisor, Martin Takáč, for all the hours spent in conversation and endless e-mail chains about everything relevant to the topic of this thesis. I am very grateful to Marek Šuppa, Ondrej Jariabka and Peter Poláčik, for patiently answering all the technical and non-technical questions I have came up with over the course of writing this thesis. Special thanks belongs to Pavol Námer and John O'Donoghue for their comments, insights and knowledge sharing that contributed to finishing this thesis, and Diana Valková for her moral support.

Abstrakt

Jeden z problémov samoorganizujúcich sa máp je výskyt mŕtvych neurónov, ktoré spôsobujú, že kapacita siete nie je plne využitá. Tento problém zväčša spôsobuje zlá náhodná inicializácia váh siete, ktorá spôsobí, že niektoré neuróny budú mať váhy ďaleko od ktoréhokoľvek vstupu. Toto spôsobí, že tieto neuróny nikdy nevyhrajú súťaž a s veľkou pravdepodobnosťou ani nebudú dostatočne adaptované vítazovou susedskou funkciou.

V tejto práci navrhneme a otestujeme niekoľkých modifikácií učiaceho algoritmu samoorganizujúcich sa máp, ktoré by mali pomôcť pri prevencii mŕtvych neurónov, a zároveň preskúmame priestor parametrov navhrnutých metód a ich vplyv na výsledky.

Kľúčové slová: mŕtve neuróny, samoorganizácia, samoorganizujúce sa mapy, neurónové siete

Abstract

One of the problems of self-organizing maps is the occurrence of so-called *dead units*, which cause the map to not be fully utilized. A bad initialization of weights can cause some neurons to have their weights far from any input data, rendering them useless, because they will never win the competition and, most likely, will not be in close proximity to the winners neighborhood to be sufficiently adapted by the neighborhood function.

In this thesis, we propose several modifications to the self-organizing maps training algorithm aimed at preventing the occurrence of dead units, along with exploring their parameter space and how the parameter values influence the outcome.

Keywords: dead units, dead neurons, self-organization, self-organizing map, SOM, neural networks

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In this thesis, we will deal with self-organizing maps. Self-organizing maps are primarily used for clustering and dimensionality reduction. Self-organizing maps (SOM) have the characteristic that neurons retain information about available data in their weights. The map size is one important parameter in SOMs. With smaller maps, each neuron must cover more of the individual input data. This has an impact on the cluster quality. For a larger map, the neurons can represent the input data more precisely. However, this has a drawback. Training a big self-organizing map should improve the cluster quality, but it has also been noted to introduce a phenomenon called dead units. This is where some of the neurons on the map are left unused.

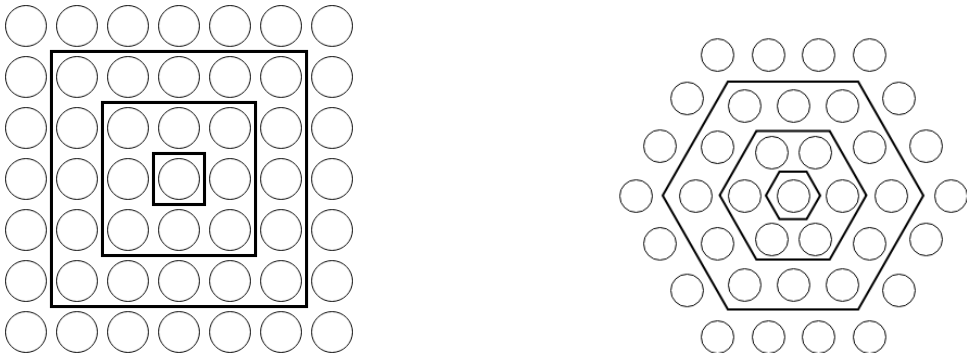
In this thesis, we will experiment with different methods for preventing dead units in self-organizing maps. The field of dead units is not very well explored, so in this paper, we will try several methods to minimize the occurrence of dead units across various parameter combinations.

The thesis is structured as follows: in the 1st chapter, we will formally and in detail describe self-organizing maps. In the 2nd chapter, we will show related work to the problem of dead units. In the 3rd chapter, we will describe the methods we will be using to reduce the number of dead units in SOM. In the 4th chapter, we will detail the results of these methods and compare them. The final chapter will be the discussion.

1.2 Self-organizing map

A self-organizing map is a model of neural networks authored by Tuevo Kohonen [3]. It is an unsupervised learning method and thus can be appropriate for dealing with problems where the data is unlabeled. In other words, the algorithm cannot use the information of output neurons during training.

A useful property of self-organizing maps is that they are capable of preserving the topological properties of the input space. This makes it easy to visualize the characteristic features of the input space later. For the map itself, neurons are typically arranged in a one-dimensional structure or a two-dimensional structure in a rectangular or hexagonal grid (Figure 1.1). This neuron arrangement represents the d-dimensional input space in which the distance between two neurons is usually given by the Euclidean distance of the underlying coordinate vectors (Equation 1.1). Two patterns similar to one another induce responses from neurons that are physically close to each other in output space also.



(a) Rectangular grid. (b) Hexagonal grid.

Figure 1.1: Types of neuron arrangements in SOM.

1.2.1 Biological inspiration

One well known representation of this type of feature mapping is in biological neural networks - in particular, the human brain. Topological maps are present in many parts of brain, mostly in the cerebral cortex. These maps are not fully developed at birth and undergo change in the early stages of growth. These changes occur when the brain receives outside stimuli, e.g. such as the brain receiving audio and visual inputs. They turn out to be an effective way to represent important features from any input stimuli.

A parallel can be drawn between this process and self-organization during unsupervised learning and thus, it inspired efforts to create the computational models of this mapping. This spawned many efforts to simulate the process of self-organization seen in the brain.

One of the earliest examples of these biologically inspired models was the Willshaw-von der Malsburg model[4]. This model attempted to understand the mechanical projection from the retina to the cerebral cortex.

1.2.2 Kohonen model

The Kohonen model is another. It is computationally simpler than the Willshaw-von der Malsburg model. In the Kohonen model, a neighbourhood function (Equation 1.4) is used and the input is represented as n-dimensional vectors. The learning algorithm consists of two steps which are repeated on each input datum (randomly selected from the training set).

- Competition (find winner neuron)
- Learning (update weights of winner neuron and its neighbourhood)

The competition phase refers to the form of unsupervised learning used in this model. Neurons *compete* with one another with the winner being chosen to respond to the input data. The winner of each competition step is known as the winner neuron (or best matching unit - BMU). Where Euclidian distance is used as the distance metric for this algorithm, the Euclidean distance is calculated between each neuron and the input datum.

$$d_E(\vec{x}, \vec{w}_i) = ||\vec{x} - \vec{w}_i|| \tag{1.1}$$

The winner neuron has weights which are closest (the most similar) to the input datum:

$$i^* = arg\, min_i(d_E(\vec{x}, \vec{w}_i)) \tag{1.2}$$

Following the competition step, the *learning* phase begins. In this step, the neuron weights are adapted. Adaptation here means that neurons which are topologically close to the winner have their weights adapted towards the winner neuron:

$$\Delta\vec{w}_i = \alpha \cdot (\vec{x} - \vec{w}_i) \cdot h(i, i^*) \tag{1.3}$$

α is the learning rate, which decreases with time. The neighbourhood function (Figure 1.2) represents the radius around the winner neuron of which other neurons are adapted, i.e. defines how many neurons within the winner neuron's neighbourhood will be adapted.

$$h(i, i^*) = e^{-\frac{{d_E}^2(i,i^*)}{\sigma^2(t)}} \tag{1.4}$$

Where $d_E(i, i*)$ represents the Euclidean distance between the lattice coordinates of neuron i and winner neuron i^*.

The neighbourhood size is represented by σ. σ decays during training, decreasing the size of the neighbourhood with each learning step.

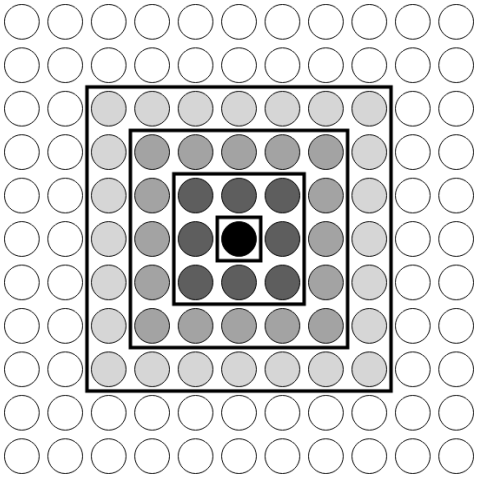


Figure 1.2: Neighbourhood of a given winner neuron. The center black point represents the winner neuron and concentric squares around it indicate the decaying weight of adaptation applied to neighbouring neurons.

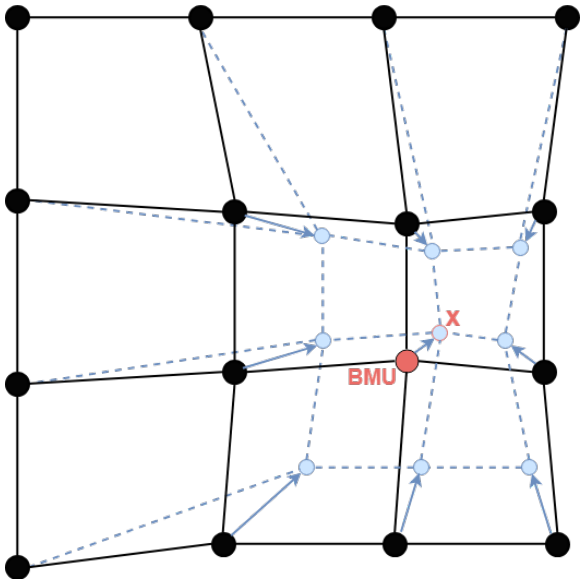


Figure 1.3: Updating the best matching unit and neurons in its neighbourhood towards input sample $\vec{x}$. The black circles represent the map before update and the blue circles represent the map after update.

Initialization

Before the competition and learning steps can begin, the map must be initialized. This is a well known problem for all iterative methods of learning neural networks. Kohonen used a random initialization of SOM weights [5], and this is the standard way to initialize SOM. We will do the same in this thesis.

Typically this random initialization is done with reference to the range of the input data. This means that if all the input data are in the range (0, 1), the neuron weights will be generated randomly from this range.

1.3 Dead units

One of the problems faced in training self-organizing maps is the occurrence of "dead" units. Dead units are neurons whose randomly initialized weight vectors are so far from any data point, that they never get chosen as winner neurons (or BMUs), nor are they in the close neighbourhood of another BMU, hence they are never sufficiently adapted to move closer to data. These dead units cause the network to not be fully utilized.

1.3.1 Causes

Dead neurons are usually caused by badly initialized weights in the SOM. Some neurons will have weights far from the input data, therefore they will not be adapted at all or too little, if they are at the edge of the winner neuron's neighbourhood.

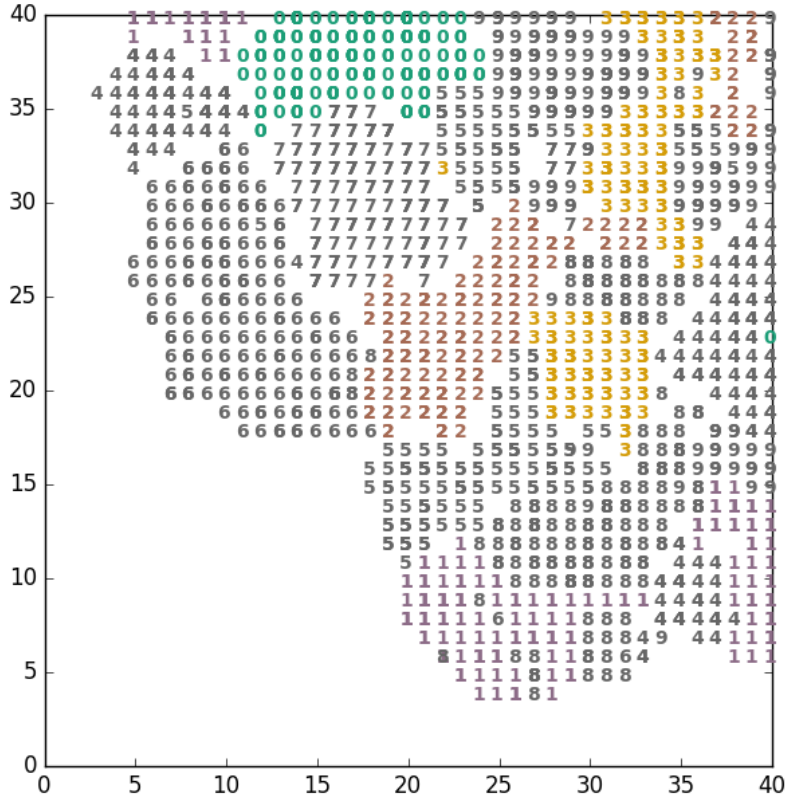


Figure 1.4: Although ignored for training of SOMs, the MNIST dataset does include labels for each observation. These are plotted for neurons which have been activated while the empty spaces in the topology represent the dead neurons. This SOM of size 40x40 has ∼38% dead units.

Chapter 2

Related work

There are not many works focusing specifically on the problem of dead units. Most works mentioning dead units deal with them specifically for their own agenda and only briefly mention the method used. The following papers show methods that adjust parameters according to the input space and we can see that dead units are closely related to the neighbourhood size and other SOM parameters.

2.1 Parameter-Less Self-Organizing Map (PLSOM)

In 2003 Berglund and Sitte proposed the parameters and update function should depend on the maps conditions instead of external variables, like learning rate. As internal condition for scaling these variables, they selected the Euclidean distance from the current input to the closest weight vector, normalized (divided) by the maximum Euclidean distance from any input seen so far to its closest weight vector. If the normalized Euclidean distance is large, the map needs to change more to fit future inputs. If it is small, the fit is good already and map does not need to change much. The idea behind PLSOM is that parameters should not be determined by the number of epoch, but by how good the topological representation of the input state is [1].

2.1.1 Algorithm

ϵ is the normalized Euclidean distance - a scaling variable depending on how good the fit of the weight vector of the winner neuron is to the last input.

$$\epsilon = \frac{||\vec{x} - \vec{w}_{i^*}||^2}{\rho} \tag{2.1}$$

$$\rho(t) = max(||\vec{x}(t) - \vec{w}_{i^*}(t)||^2, \rho(t-1)) \tag{2.2}$$

$$\rho(0) = ||\vec{x}(0) - \vec{w}_{i^*}(0)||^2 \tag{2.3}$$

$\epsilon(t)$ represents the Euclidean distance between the input datum and the winner neuron at time t. If ϵ is large, the map needs to readjust more, since it is not fitting the data well and when ϵ is small, the input already fits the map well. In the PLSOM algorithm, traditional annealing of the neighbourhood is replaced with

$$\beta(t) = constant \ \forall t \tag{2.4}$$

because the neighbourhood size is determined by ϵ [1]. The equation for the neighbourhood function is defined as

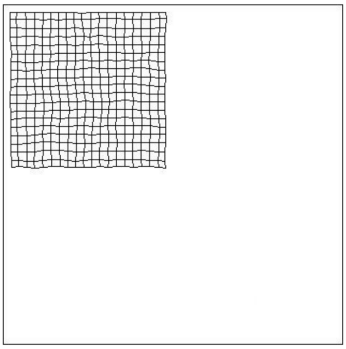
$$h(i, i^*)(t) = e^{-\frac{{d_E}^2(i,i^*)}{\sigma^2(t)}} \tag{2.5}$$

where $d_E(i, i^*)$ is the Euclidean distance between neurons i and i^*. The equation for the weight updates is defined as

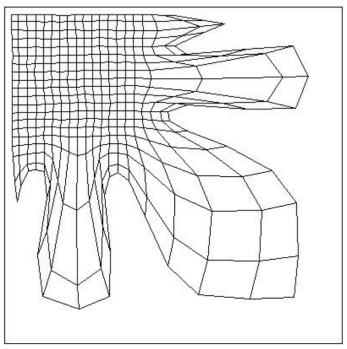
$$\vec{w}_i(t+1) = \vec{w}_i(t) + \epsilon(t) \cdot (\vec{x}(t) - \vec{w}_i(t)) \cdot h(i, i^*)(t) \tag{2.6}$$

where $\vec{w}_i(t)$ is the weight vector of neuron i at the time t, ϵ is the normalized Euclidean distance at the time t and $h(i, i^*)(t)$ is the neighbourhood function at the time t.

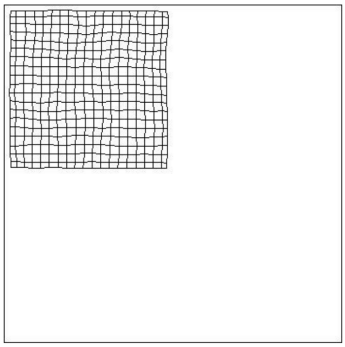
The learning rate has been eliminated and replaced by $\epsilon(t)$. In Figure 2.1 is the performance comparison of the standard SOM algorithm, PLSOM and Matlab-implemented SOM algorithm from the paper.



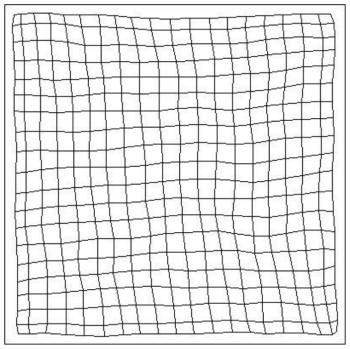
(a) SOM weight vector position in the input space after training for 50,000 epochs with uniformly distributed pseudo-random 2-dimensional input, ranging from 0 to 0.5.



(b) Same SOM as in figure (a) after 20,000 further training steps with inputs ranging from 0 to 1.0.



(c) PLSOM weight vector position in the input space after training for 50,000 epochs with uniformly distributed pseudo-random 2-dimensional input, ranging from 0 to 0.5.



(d) Same PLSOM as in figure (c) after 20,000 further training steps with inputs ranging from 0 to 1.0. Note the difference between this figure and figure (b).

Figure 2.1: Visual comparison of SOM and PLSOM [1].

In the picture above you can see visual comparison between SOM and PLSOM. Normal SOM fails to utilize most of the neurons after 50,000 epochs whereas PLSOM utilizes all of them after just 20,000 epochs. The PLSOMs topological map also looks similar to input, whereas topological map of standard SOM does not.

2.2 Improved PLSOM (PLSOM2)

The earlier modification, PLSOM, solved some problems of SOM, but introduced other. One of them is overreaction to extreme outliers, which happens even when the SOM trains for a longer time period. This causes problems, since any standard dataset of acceptable size will most probably have outliers. Another problem PLSOM introduced is that the initial weights can affect the amount of adaptation if the weights are initialized too far from the input space, causing training examples to cluster on the edges. PLSOM2, introduced in [2], tries to fix these problems. PLSOM2, instead of using the size of the error relative to the maximum error to change the scaling of the update, uses the range of observed inputs.

Even though the PLSOM was an overall improvement over standard SOM, the success of the SOM is heavily dependent on initial weights and the number of outliers in the dataset. The scaling of the update in PLSOM is based on $err(t)$ relative to the largest error seen up to that point. When weights are initialized far from input space, the initial error ($err(0)$) is very large and all further adaptations will be very small, causing the map not to be ordered at all and overreact to extreme outliers. PLSOM2 does not scale error as PLSOM does, instead scales errors relative to the diameter of the union of observed inputs [2]. This means that no feature map is stored, therefore the initial weight distribution does not get to influence the map after several epochs.

2.2.1 Algorithm

The algorithm performs two main steps for each input. The first step is determining the size of the input space (S), the second one is updating the weights. The weight update depends on the input space, since these operations run sequentially. The input space size is defined as the diameter of the dataset:

$$S_t = \max_{i,j}(||\vec{x}_i - \vec{x}_j||^2), \quad \forall i, j \leq t \tag{2.7}$$

where $x_i \in \mathbb{R}^n$ is the input at time i and t is the current time. The weight update is defined as:

$$d(t) = \min\left(\frac{err(t)}{S}, 1\right) \tag{2.8}$$

where $d(t)$ is a scaling variable representing the fit and S is the input space and $err(t)$ is error represented by the Euclidean distance between input datum and weight vector of the winner neuron. The neighbourhood size is determined by $d(t)$ and

$$\Theta(d(t)) = \beta \ln(1 + d(t) \cdot (e - 1)) \tag{2.9}$$

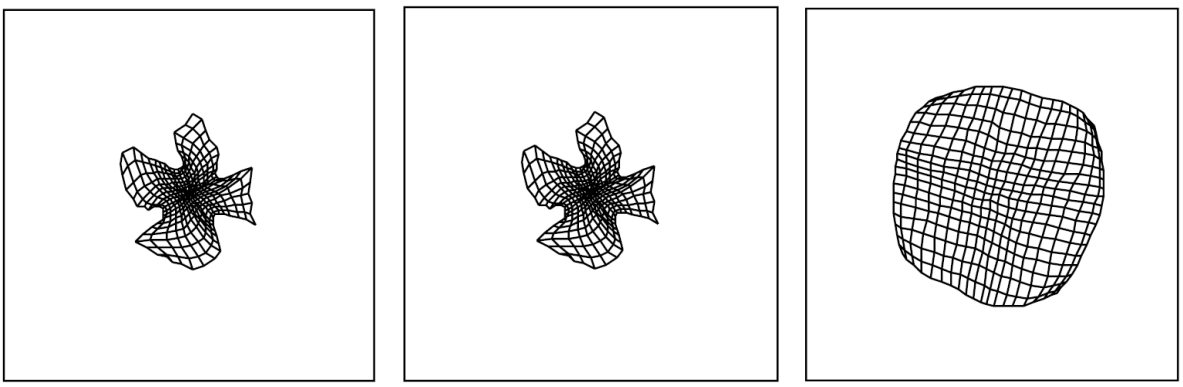
where $(e-1)$ is the scaling factor, which is chosen to ensure the range of $d(t) = [0, 1]$ maps into the range of $\Theta(d(t)) = [0, \beta]$ and $\beta =$ constant $\forall t$ is neighbourhood range. The value of Θ is then used in the neighbourhood function:

$$h(i, i^*)(t) = e^{-\frac{{d_E}^2(i,i^*)}{\Theta^2(d(t))}} \tag{2.10}$$

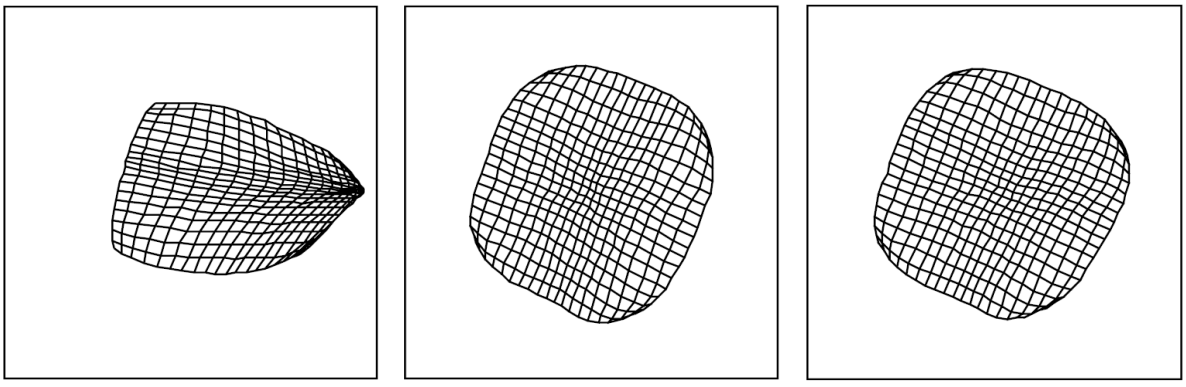
where $d_E(i, i^*)$ is the Euclidean distance in output space between the winner neuron i^* and neuron i, which is the neuron currently being updated. This gives a value for $h(i, i^*)(t)$ that decreases with increased distance from i^*, and the rate of decrease is determined by $d(t)$. The weight update functions are:

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \Delta\vec{w}_i(t) \tag{2.11}$$

$$\Delta\vec{w}_i(t) = d(t) \cdot h(i, i^*)(t) \cdot (\vec{x}(t) - \vec{w}_i(t)) \tag{2.12}$$



(e) SOM before outlier. (f) SOM after outlier. (g) PLSOM before outlier.



(h) PLSOM after outlier. (i) PLSOM2 before outlier. (j) PLSOM2 after outlier.

Figure 2.2: Comparison of response to outlier of SOM, PLSOM and PLSOM2 from [2] after 200,000 epochs. Normal SOM fails to utilize most of the neurons. PLSOM overreacts to the outlier, forcing drastic topology change, whilst PLSOM2 handles the appearance of an outlier the best.

Chapter 3

Method

A standard SOM presents two types of problems: convergence and dead units. A well initialized SOM might already fit the data well, therefore saving computational time as a result. A bad initialization can cause some neurons to never be adapted or be adapted insignificantly, resulting in the map not being fully utilized, therefore degrading the quality of data representation. These neurons are called dead units.

To get a better idea how to reduce the occurrence of dead units in self-organizing maps, we created a set of experiments.

3.1 General setup

Three different datasets representing three different types of input will be tested on three different sizes of SOM: 10x10, 20x20 and 40x40. The model will be trained for 30 epochs. Each epoch refers to one training iteration in which the weights are updated according to each of the input vectors. Preliminary experiments showed that the error does not drop significantly after 30 epochs, therefore this number was chosen as a limit for other experiments in this thesis. In each epoch, the SOM will be trained (iteratively) on all data samples. Between each epoch the dataset is shuffled such that the order in which the dataset is iterated through will not have a significant impact on the SOM. All input data will be normalized by:

```
data = data / data.argmax()
```

3.1.1 MNIST

The digits dataset[1] which is made up of 1797 hand written 8x8 images of digits will represent a multidimensional input with 10 classes (each digit) with a relatively uniform

[1] http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html

distribution of the classes.

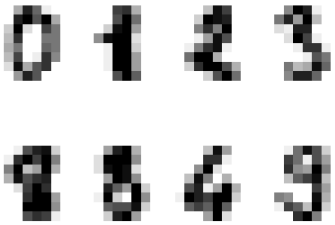


Figure 3.1: Example from the digits dataset[2].

3.1.2 RASTER

This dataset is created by a simple script using two for cycles and numbers from range (0, 1) with a step of 0.01. Resulting dataset with 9801 samples is then randomly shuffled. We use a subset with 2000 samples to train our experiments.

```
[[0.07, 0.09],
 [0.03, 0.14],
 [0.21, 0.25],
 ...,
 [0.51, 0.74],
 [0.42, 0.46],
 [0.54, 0.31]]
```

Figure 3.2: Example of our raster dataset.

This dataset will be used as a low-dimensional input space (with no internal structure) for our experiments.

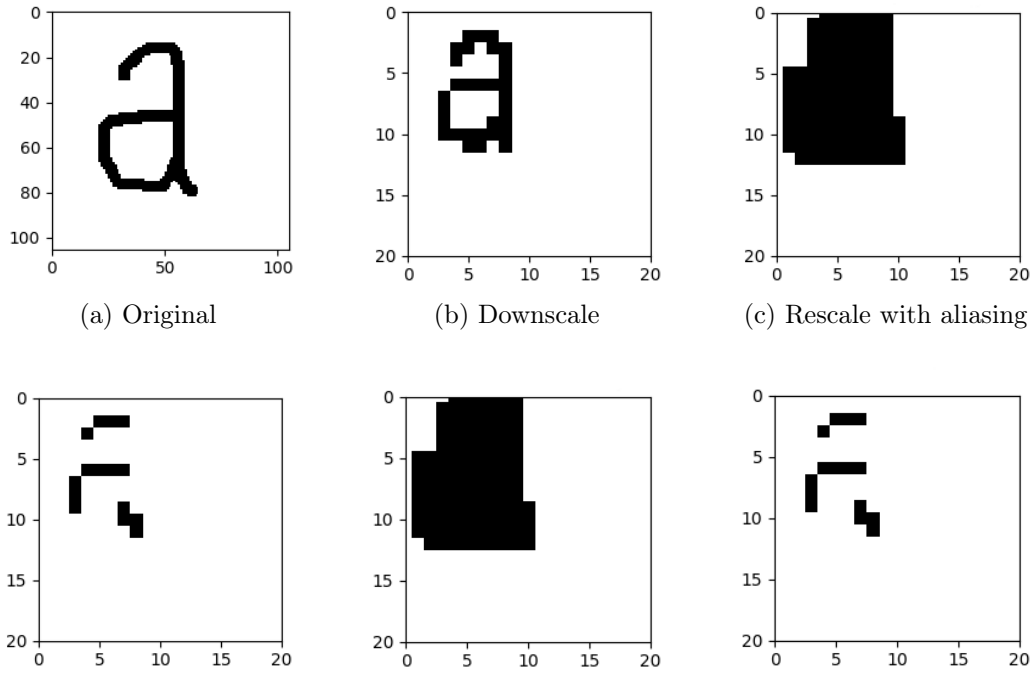
3.1.3 OMNIGLOT

OMNIGLOT is dataset of 1623 different handwritten characters from 50 different alphabets. Each of the 1623 characters was drawn online via Amazon's Mechanical Turk by 20 different people [6], that means the whole dataset has 32460 samples. This dataset therefore represents a multidimensional input in which there are many classes having few samples per class. Each image is 105x105 pixels.

[2]https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

Figure 3.3: Example of the OMNIGLOT dataset[3].

In order to avoid computational restriction, we had to come up with a way to reduce the dataset that will be trained on our experiments. Three methods were tested to achieve this - resizing (with and without aliasing), rescaling (with and without aliasing) and downscaling the images. Preliminary experiments, Figures 3.4 and 3.5, have shown that downscaling to 7x7 image preserves most of the information from the original image. Therefore this method was initially chosen.

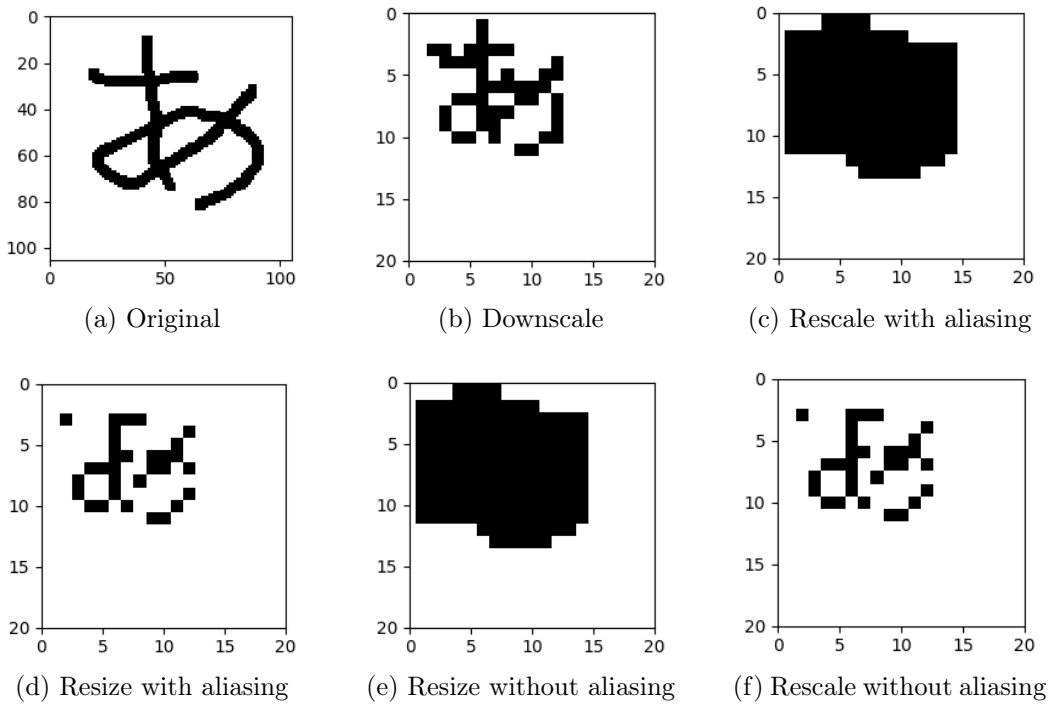


(a) Original (b) Downscale (c) Rescale with aliasing

(d) Resize with aliasing (e) Resize without aliasing (f) Rescale without aliasing

Figure 3.4: Results of different image size-reducing methods for (Cyrillic alphabet).

[3]https://github.com/brendenlake/omniglot/blob/master/omniglot_grid.jpg



(a) Original (b) Downscale (c) Rescale with aliasing

(d) Resize with aliasing (e) Resize without aliasing (f) Rescale without aliasing

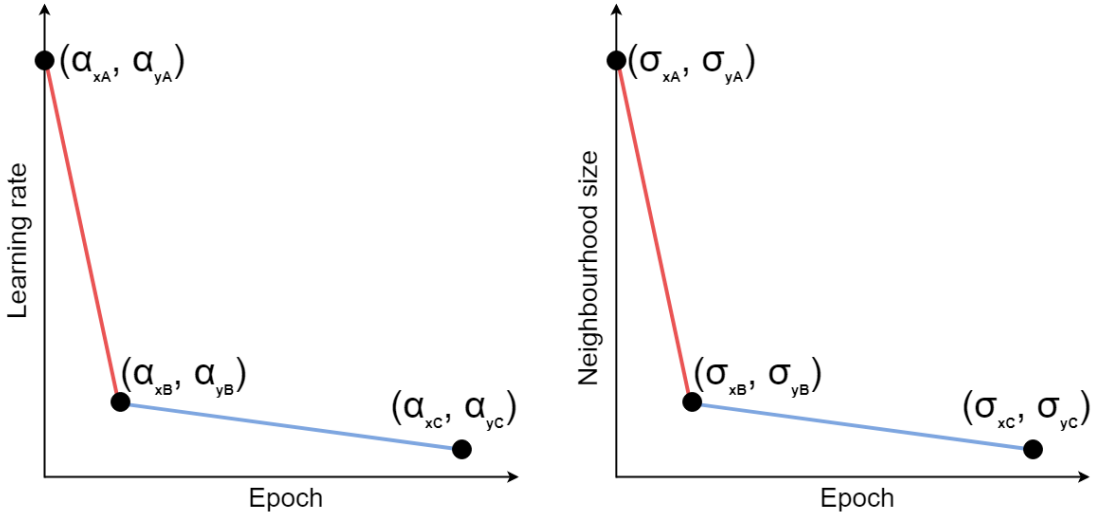
Figure 3.5: Results of different image size-reducing methods for (Japanese alphabet).

Using this downscaled dataset, however, did not provide the gains expected. Despite being able to train with more samples, downscaling led to poorer training performance than using the original data with a smaller subset of the data. Downscaling to 15x15 images lead to similar results. It may be that the information loss which occurs in this process is more significant than originally thought and thus we ended up using the data in its original format instead. We ended up using less characters in full resolution from each alphabet instead.

For the purposes of this thesis, we will be using a subset of 30 different alphabets with one character (20 samples each) from each alphabet, i.e. 600 samples in total.

3.2 Standard self-organizing map

This method represents a standard approach to training self-organizing maps[3] and will be used as a benchmark to other methods. Most methods utilize parameter annealing. The first phase of annealing, called the **initial organization phase**, is to start off with a bigger neighbourhood and quickly, within few epochs, get to a small neighbourhood size. For a neighbourhood function, we use the Gaussian neighbourhood (Equation 1.4). σ will be used in further text as a parameter representing neighbourhood size. Bigger values of σ mean bigger neighbourhood sizes of the winner neuron i^*. The second phase, called the **fine-tuning phase**, is used for fine-tuning the neuron weights.



(a) Annealing for learning rate

(b) Annealing for neighbourhood size

Figure 3.6: Annealing.

In Figure 3.6, the red line represents the initial organization phase and the blue line represents the fine-tuning phase. xA is the first epoch and will always be 0, σ_{yA} (α_{yA}) is the initial neighbourhood size (and initial learning rate). xB is the number of epochs after which the breaking point occurs and σ_{yB} (α_{yB}) is the neighbourhood size (or learning rate) at this point. The breaking point is a parameter (specified prior to running the SOM algorithm) and denotes the epoch in which the SOM should transition from the initial organization phase to the fine-tuning phase. xC is always equal to maximum number of epochs. σ_{yC} (α_{yC}) is the value of the neighbourhood size (or learning rate) at the end of training.

In our experiment, we fixed the following parameters for the learning rate:

$$\alpha_{yA} = 1.0 \tag{3.1}$$

$$\alpha_{yC} = 0.1 \tag{3.2}$$

and for neighbourhood size:

$$\sigma_{yB} = 1.0 \tag{3.3}$$

$$\sigma_{yC} = 0.5 \tag{3.4}$$

Both α_{yA}, σ_{xA} are 0 and α_{xB}, σ_{xB} are set to the breaking point. Learning rate is calculated with below function:

$$\alpha_{new} = \alpha_{old} + \frac{(t - \alpha_{xA}) \cdot (\alpha_{yB} - \alpha_{yA})}{(\alpha_{xB} - \alpha_{xA})} \tag{3.5}$$

Neighbourhood size is calculated with below function:

$$\sigma_{new} = \sigma_{old} + \frac{(t - \sigma_{xA}) \cdot (\sigma_{yB} - \sigma_{yA})}{(\sigma_{xB} - \sigma_{xA})} \tag{3.6}$$

where t is the number of sample. The formula represents linear decay from (xA, yA) to (xB, yB). All combinations of α_{xB}, α_{yB}, σ_{xB}, σ_{yA} shown below were explored.

Table 3.1: Parameter values used in experiments.

α_{xB}, σ_{xB}	1, 3, 7, 12, 20
α_{yB}	0.1, 0.2, 0.5
σ_{yA}	2.0, 5.0, 10.0, 20.0, 50.0, 100.0

Our hypothesis is that starting with larger neighbourhood sizes can help minimize the number of dead units. We also explore the parameter space of the learning rate, because annealing the learning rate is the usual way of SOM training, but our main focus is on the influence of the neighbourhood size annealing scheme on the number of dead units.

3.3 Training random dead unit

In [7], the authors create an algorithm with two steps to prevent the occurrence of dead units. The algorithm starts off with a direct adaptation: the SOM is trained on an input sample and the winning neuron and its neighbourhood adapts as per standard SOM. The proposed *new* step, *inverted adaptation*, randomly picks a dead unit, finds the closest input sample for the chosen neuron and adapts its weights towards the chosen input sample. The hypothesis is that this creates the possibility that outliers which would never have been adapted otherwise will now be adapted.

The authors, however, do not explore how much the method affects the occurrence of dead units; instead they focus on using this method to represent a 3D space in 2D. This is why we decided to re-implement this method, explore it deeper and compare to other methods.

Our method is a modified version of the method proposed in [7]. In standard training steps, we train the SOM on the dataset, whilst keeping a map of all neurons that have not been trained yet. After each standard step, we randomly choose a dead unit, find the closest input sample and then adapt its weights and the weights of neurons within the winner neurons neighbourhood with this input sample.

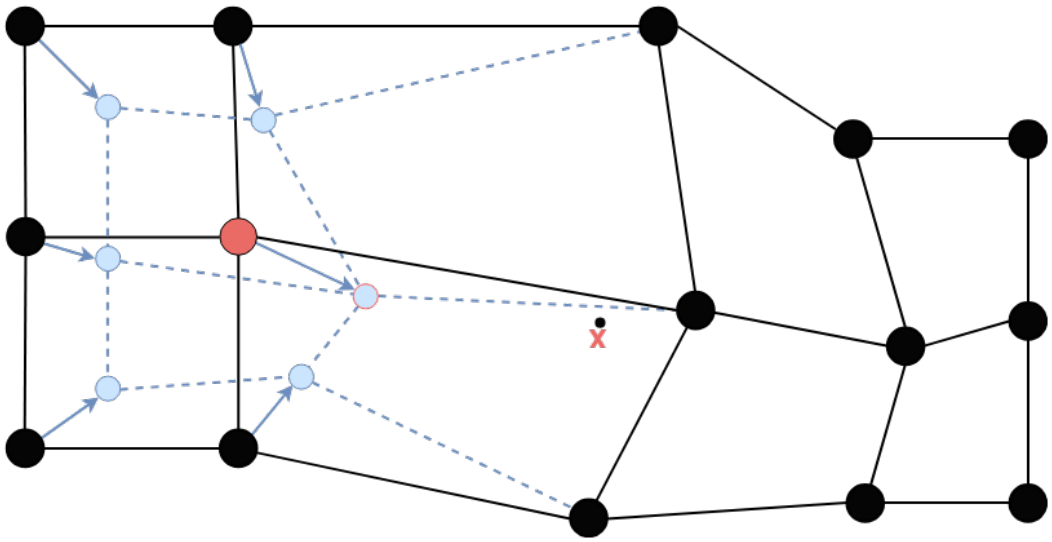


Figure 3.7: Updating randomly chosen dead unit (red dot) and its neighbourhood towards its closest input sample (x). Black circles represent the map (weight vectors as points in space) before update and the blue circles represent the map after update.

For the purposes of this thesis, the dead unit is left in the list of dead units, so it can be selected again from this list. This is done because the dead unit is only nudged in the right direction in the extra step and did not win the actual competition, thus leaving it marked as a dead unit until it has been updated as part of the standard step (either being a BMU or being in the close neighbourhood of a BMU). The nudge does not guarantee that the unit has been moved/adapted enough to be able to win a regular competition - hence it will only be removed from the set of dead units when it does.

3.4 Training dead units for novel inputs

The basic idea of this method is that if the winner neuron has its weights similar enough to the input sample, the input is most probably an instance of the same category and the winner can be adapted.

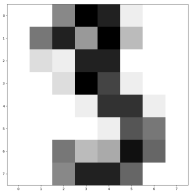
If not, rather than updating the winner neuron and it's neighbourhood, a different *unused* neuron is chosen. This neuron and its neighbourhood are updated instead. When this happens, it is considered a *novel* input. A threshold value must be set to evaluate whether the Euclidean distance between the winner neuron and the input vector can be considered to be in the same category. The problem can be formalized as:

if $d(\overrightarrow{input}, \overrightarrow{BMU}) >$ threshold:
    ignore BMU
for each dead_unit:
    calculate $d(\overrightarrow{input}, \overrightarrow{dead\_unit})$
update dead_unit with $min(d_E(\overrightarrow{input}, \overrightarrow{dead\_unit}))$

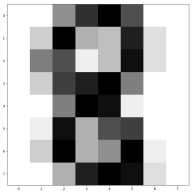
where the distances, $d_E(\overrightarrow{a}, \overrightarrow{b})$, represent the Euclidean distance between two vectors.

We quickly faced the problem of data separability, which is the property of the data we are using to train the SOM. Numeric characters like 3 and 8 are similar enough to confuse the network for example.

Thus we experimentally measured Euclidean distances within and across some categories. For the MNIST dataset, we chose digits 3 and 8 to analyze (Figure 3.8).



(a) Sample digit three taken from MNIST dataset



(b) Sample digit eight taken from MNIST dataset

Figure 3.8: Comparison between digits three and eight.

The mean value of the Euclidean distances within the sample of *3 digits* was 0.45 and with the sample of *8 digits* was 0.49. The same value when comparing *3s* with *8s* was 0.58. We therefore conject that a reasonable threshold value to experiment with will be in the approximate range 0.4-0.6. Thus values of 0.4, 0.5, and 0.6 were chosen.

Data separability was also an issue in the Omniglot dataset. The most similar letters - *ayb* from the Armenian alphabet and *dha* from the Balinese alphabet - were chosen to analyze the separability (Figure 3.9).

(a) Letter *ayb* from Armenian alphabet taken from Omniglot dataset

(b) Letter *dha* from Balinese alphabet taken from Omniglot dataset

Figure 3.9: Similarity comparison between letters from Armenian and Balinese alphabets.

Here the mean value of the Euclidean distances between each *ayb* letter was 37.4 and between *dha* letters this was 36.9. The minimum, mean and maximum value of the Euclidean distances between letters *ayb* and *dha* was 30.5, 37.67 and 45.6, respectively. Therefore reasonable threshold values to experiment with were posited to be 30, 40 and 50.

The raster dataset has no obvious categories like the other two datasets, so data separability is not an issue. Therefore the same method could not be used to decide on a suitable threshold value range. Instead, the closest distance between two samples was calculated - the value being 0.01. So to force neurons into clusters, threshold values of 0.02, 0.05 and 0.1 were chosen for experimentation.

The results are discussed in more detail in 4.

3.5 Computational restrictions

This section details the computational restrictions which were faced in the course of our experiments. All experiments were carried out on a personal laptop (Table 3.2). Whist we did have access to a shared server with greater resources, this did not help as the code is single threaded and so could not utilize the full available resources. Experimentation with this system did not bring any gains in terms of run-time.

A potential improvement would be to rewrite the code or using Apache Spark to parallelize some of the more expensive operations.

For example updating the neuron weights at each step before finding the BMU could be handled in parallel, significantly reducing overall compute time.

Table 3.2: Laptop specifications.

Processor	i5-4210U at 1.7 GHz
Memory	12 GB DDR3L-SDRAM at 1600 MHz
Graphics	Intel® HD Graphics 4400
Storage	5400 RPM SSHD

The below table shows how many parameter combinations are considered for each method. Some gains in compute time were made by running each combination on a separate thread (up to 3 at a time).

Table 3.3: Counts of experiment parameters. Exact parameters can be seen in Table 3.1. α_{xB} and σ_{xB} are the breaking points, α_{yB} is learning rate at breaking point and σ_{yA} is initial neighbourhood size.

Experiment	α_{xB}, σ_{xB}	α_{yB}	σ_{yA}	Thresholds	Combinations
Standard SOM	3	5	6	-	90
Random dead unit	3	5	6	-	90
Dead units for novel inputs	3	5	6	3	270

The following sections detail the total run times taken for each dataset, technique and map size using the system specifications given in Table 3.2.

3.5.1 MNIST

Table 3.4: Experiments on MNIST dataset using *standard SOM* method.

Map size	Time per combination	Total time
10x10	$\sim$ 2 minutes	$\sim$ 3 hours
20x20	$\sim$ 8.5 minutes	$\sim$ 13 hours
40x40	$\sim$ 35 minutes	$\sim$ 52.5 hours

Table 3.5: Experiments on MNIST dataset using *training random dead unit* method.

Map size	Time per combination	Total time
10x10	$\sim$ 2.5 minutes	$\sim$ 4 hours
20x20	$\sim$ 9.5 minutes	$\sim$ 14 hours
40x40	$\sim$ 38.5 minutes	$\sim$ 58 hours

Table 3.6: Experiments on MNIST dataset using *training dead units for novel inputs* method.

Map size	Time per combination	Total time
10x10	∼ 2 minutes	∼ 9 hours
20x20	∼ 9 minutes	∼ 40.5 hours
40x40	∼ 36 minutes	∼ 162 hours

3.5.2 RASTER

Table 3.7: Experiments on RASTER dataset using *standard SOM* method.

Map size	Time per combination	Total time
10x10	∼ 2 minutes	∼ 3 hours
20x20	∼ 7.5 minutes	∼ 11 hours
40x40	∼ 33 minutes	∼ 49.5 hours

Table 3.8: Experiments on RASTER dataset using *training random dead unit* method.

Map size	Time per combination	Total time
10x10	∼ 3 minutes	∼ 4.5 hours
20x20	∼ 9 minutes	∼ 13.5 hours
40x40	∼ 41 minutes	∼ 61.5 hours

Table 3.9: Experiments on RASTER dataset using *training dead units for novel inputs* method.

Map size	Time per combination	Total time
10x10	∼ 2.5 minutes	∼ 11 hours
20x20	∼ 8 minutes	∼ 36 hours
40x40	∼ 36 minutes	∼ 162 hours

3.5.3 OMNIGLOT

Table 3.10: Experiments on OMNIGLOT dataset using *standard SOM* method.

Map size	Time per combination	Total time
10x10	∼ 4 minutes	∼ 6 hours
20x20	∼ 19 minutes	∼ 28.5 hours
40x40	∼ 89 minutes	∼ 133.5 hours

Table 3.11: Experiments on OMNIGLOT dataset using *training random dead unit* method.

Map size	Time per combination	Total time
10x10	$\sim$ 4 minutes	$\sim$ 6 hours
20x20	$\sim$ 21.5 minutes	$\sim$ 32 hours
40x40	$\sim$ 114.5 minutes	$\sim$ 172 hours

Table 3.12: Experiments on OMNIGLOT dataset using *training dead units for novel inputs* method.

Map size	Time per combination	Total time
10x10	$\sim$ 4 minutes	$\sim$ 18 hours
20x20	$\sim$ 20 minutes	$\sim$ 90 hours
40x40	$\sim$ 107 minutes	$\sim$ 481.5 hours

3.5.4 Evaluation of expected runtimes for each method

The SOM has three input parameters: SOM size N, input dimension d and training set size M. To update the weights for each standard step, we need $N \cdot d$ space and $O(d)$ for the Euclidean distance and $O(N)$ to find the winner, by comparing the input sample with the weights of each neuron. This gives us time complexity $O(MdN)$ for each epoch.

In addition to the standard training step, the second method also needs to choose a random dead unit ($O(1)$) and compare it to each input sample in the training set ($O(M)$), times $O(d)$ for each Euclidean distance calculated.
This method then gives us $O(M \cdot (dN + dM)) = O((MdN) + M^2 d)$ instead of $O(MdN)$ in the first method. ThZere is an extra term in the second equation, which is quadratic in M. Therefore, before using this method, it should be considered on what data it is going to be used. If there is going to be a lot of data (M), it might not be the wisest choice, but it should run well on big vectors (d), e.g. camera images, or self-organizing maps with big sizes (N).

The third method compares the Euclidean distance between the input and the found winner to a threshold in each epoch and, in the case of a novelty input, it compares this input sample to each dead unit from the set of dead units. The size of the dead units set can be N at most. This gives us (in addition to $O(Nd)$ for the standard case) another at most $O(Nd)$ operations in cases of novelties. Thus the third method still runs in $O(MdN)$ time.

This means the second method is slower than both first and third method and should be used with caution in the case of large data sets.

3.6 Implementation

3.6.1 Programming language

We used Python as our primary programming language. We chose this language because of its ease of use and the large variety of libraries supporting machine learning and neural network training.

3.6.2 Libraries

We used a variety of libraries. The most important are listed below.

NumPy

NumPy[4] is a library for Python, supporting large, multi-dimensional arrays and matrices and a large collection of high-level mathematical functions to operate on these arrays.

We will be using commonly used functions like *exp*, *power*, *subtract*, *dot*, *transpose* to work with vectors and matrices, as well as more complex ones such as *nditer* and *unravel_index*.

scikit-learn

Scikit-learn[5] is a machine learning library for Python. We will be using scikit-learn mainly for its built in MNIST dataset and its *shuffle* function to shuffle our datasets in a consistent way. To experiment with resizing, rescaling and downscaling OMNIGLOT images, we will be using *scikit-image* for its image processing capabilities.

Matplotlib

Matplotlib[6] is a Python 2D plotting library. We will be using it to plot various graphs and visualizations.

[4]http://www.numpy.org/
[5]https://scikit-learn.org/
[6]https://matplotlib.org/

3.6.3 Code

In Listing 1 is train function of standard SOM. In each epoch, the algorithm goes through the whole dataset, finds BMU for each input sample and updates its weights and weights of neurons within its neighbourhood.

Listing 1: Train function of standard SOM.

```
def train(self, data):
    # Cycle through epochs and data points to train SOM.

    ep = 0
    self._t = 0

    for ep in range(self._epochs):
        data = shuffle(data)

        for sample in data:
            self.update(sample, self.winner(sample))
            self._t += 1
```

In Listing 2 is find_sample function from second experiment in which we adapt randomly chosen dead unit. This function randomly picks a dead unit from a map of dead units, then finds the closest input sample to its weights and returns it to be updated. If no neurons are left in the dead units map, update is skipped.

Listing 2: Function find_sample that finds the closest input sample to randomly chosen dead unit.

```
def find_sample(self, data):
    # Return random dead unit and data input closest to it.

    dead_units = np.transpose(np.nonzero(self._dead_units_map))
    if len(dead_units) == 0:
        return None, None
    # randomly choose dead unit from map of dead units
    winner = tuple(dead_units[np.random.randint(len(dead_units))])
    winner_weights = self._weights[winner]

    # find the closest input sample to this dead unit
    distances = {}
    for i in range(len(data)):
        distances[i] = np.subtract(data[i], winner_weights)
    sample = data[min(distances, key=distances.get)]
    return sample, winner
```

In Listing 3 is train function from second experiment. After standard step of training the whole dataset, extra step is added (Listing 2). If no dead unit is found, update does not happen.

Listing 3: Train function of second method that trains random dead unit in an extra step.

```
def train(self, data):
    # Cycle through epochs and data points to train SOM.

    ep = 0
    self._t = 0

    for ep in range(self._epochs):
        data = shuffle(data)

        for sample in data:
            self.update(sample, self.winner(sample))
            self._t += 1

        best_sample, winner = self.find_sample(data)
        if best_sample is not None and winner is not None:
            self.update_dead(best_sample, winner)
```

In Listing 4 is find_dead function from third experiment in which we train dead units for novel inputs. This function finds nearest dead unit closest to the given input sample if the winner neuron found in standard step does not satisfy the set threshold.

Listing 4: Function find_dead that finds closest dead unit to the given input sample.

```
def find_dead(self, x):
    # Find closest dead unit to the input sample.

    dead_units = np.transpose(np.nonzero(self._dead_units_map))
    if len(dead_units) == 0:
        return None

    distances = {}
    for i in range(len(dead_units)):
        du = tuple(dead_units[i])
        du_weights = self._weights[du]
        distances[i] = self._ed(x, du_weights)

    winner = dead_units[min(distances, key=distances.get)]
    return winner
```

In Listing 5 is train function from third experiment. Additional threshold condition has been added to train dead units when the current winner neurons weights are not close enough to the input sample.

Listing 5: Train function of third method that adds threshold for novelty inputs.

```
def train(self, data):
    # Cycle through epochs and data points to train SOM.

    ep = 0
    self._t = 0

    for ep in range(self._epochs):
        data = shuffle(data)

        for sample in data:
            winner = self.winner(sample)
            if self._distance_map[winner] < self._threshold:
                self.update(sample, winner)
            else:
                win = self.find_dead(sample)
                if win is None:
                    self.update(sample, winner)
                else:
                    self.update(sample, win)
            self._t += 1
```

3.7 Quality measures

For visualization of maps trained on MNIST dataset, winmaps will be used. Winmaps are topographical plots of the SOM, where each point represents a neuron and its winning digit.

To see how well map performs, quantization error will be calculated after each epoch. Quantization error is the average distance between each input sample and its best matching unit (winner neuron).

To see how well each method performs, a combination of quantization error and the final count/percentage of dead units will be used.

Chapter 4

Results

In this chapter, the results of the experiments - using the previously discussed methods, datasets and map sizes - will be detailed. The discussion will focus on whether the choice of method used will differ depending on the structure of the dataset and the map size.

In Section 4.1, we will talk about statistical concerns. Section 4.2 covers the MNIST dataset, whilst sections 4.3 and 4.4 cover the Raster and Omniglot datasets respectively.

4.1 Statistical Concerns

As stated above, the aim of this section is to show that the choice of method to use should differ depending on the dataset and map size. This can be formalized as follows:

$\mathbf{H}_0$. *The choice of method is irrelevant, in regards to dead units and quantization error elimination.*

$\mathbf{H}_A$. *There are material differences between the methods chosen (either standard SOM, training random dead units OR training dead units for novel inputs) in regards to dead units and quantization error elimination.*

It must be stated that as each experiment is only carried out on one random initialization of the SOM (due to computational restrictions), the results stated are dependent on that initialization. A more comprehensive approach would involve taking n random starts and comparing means of the groups, or using statistical tests.

4.2 MNIST

MNIST represents a multidimensional input with 10 classes (one for each digit) with a relatively uniform distribution of the classes.

In this section, results from each of the map sizes trained - 10x10, 20x20 and 40x40 - will be discussed. Each subsection will take a closer look at the performance of individual methods on this specific data design and map sizes. It will also cover analysis of the training parameters for the methods which performed the best. The quantization error of the best performing methods and their best performing parameters will then be compared. All other results can be found in Appendix 5.2.5.

4.2.1 10x10

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: 0%, 1%, 2% and >2%. Ranges are not necessary here as a 10x10 map has exactly 100 neurons, so it is not possible to have e.g. 1.5% dead units. These results can be seen in figure 4.1.

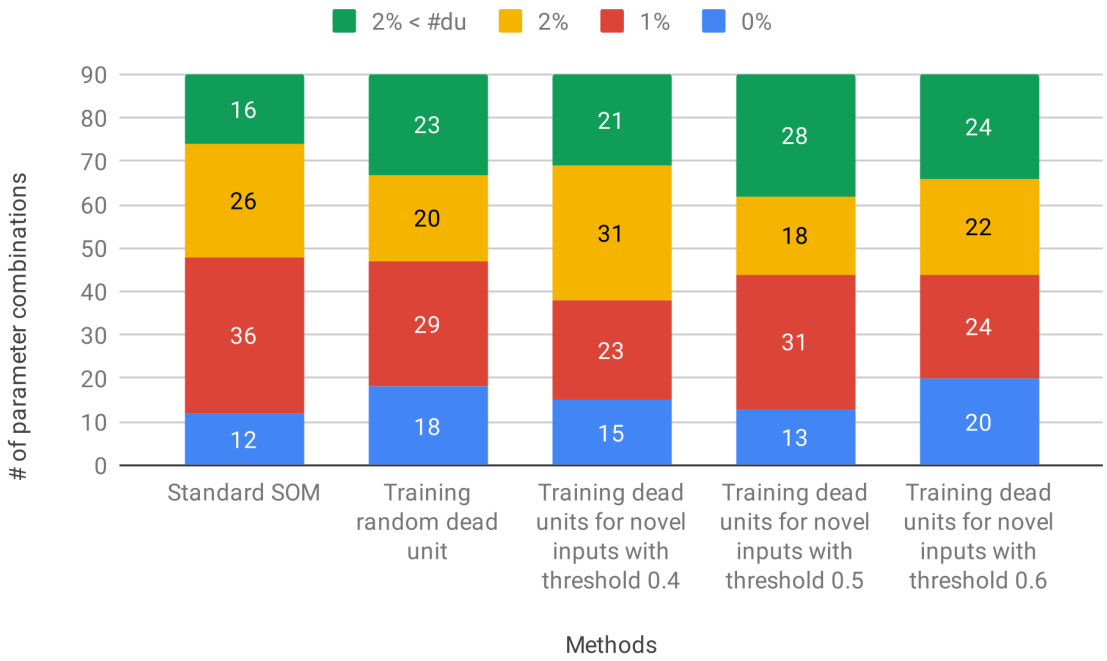


Figure 4.1: 10x10 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments. More combinations in the blue / red categories indicate those methods have a larger acceptable solution space for this problem.

Results indicate that the best performing methods for threshold 0% are **training random dead unit** and **training dead units for novel inputs with threshold 0.6**. The performance of parameter combinations for these two methods can be seen in Figure 4.2.

Training random dead unit

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	1.00%	4.00%	3.00%	0.00%	2.00%	1.00%
3	5.00%	1.00%	1.00%	1.00%	2.00%	4.00%
7	0.00%	4.00%	0.00%	4.00%	1.00%	2.00%
12	1.00%	0.00%	1.00%	1.00%	1.00%	1.00%
20	1.00%	0.00%	1.00%	2.00%	0.00%	2.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	3.00%	3.00%	1.00%	0.00%	0.00%	4.00%
3	4.00%	2.00%	2.00%	2.00%	0.00%	4.00%
7	1.00%	1.00%	1.00%	2.00%	2.00%	1.00%
12	0.00%	1.00%	2.00%	1.00%	3.00%	0.00%
20	0.00%	4.00%	3.00%	1.00%	3.00%	1.00%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	0.00%	3.00%	1.00%	2.00%	3.00%	2.00%
3	2.00%	2.00%	3.00%	1.00%	2.00%	4.00%
7	2.00%	1.00%	0.00%	3.00%	2.00%	3.00%
12	0.00%	1.00%	2.00%	3.00%	2.00%	1.00%
20	0.00%	0.00%	0.00%	1.00%	1.00%	3.00%

Training dead units for novel inputs with threshold 0.6

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	2.00%	0.00%	1.00%	1.00%	0.00%	1.00%
3	3.00%	3.00%	0.00%	1.00%	4.00%	1.00%
7	0.00%	2.00%	4.00%	3.00%	2.00%	2.00%
12	0.00%	1.00%	0.00%	2.00%	3.00%	1.00%
20	1.00%	2.00%	0.00%	2.00%	1.00%	3.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	3.00%	5.00%	2.00%	3.00%	2.00%	2.00%
3	0.00%	1.00%	2.00%	1.00%	1.00%	4.00%
7	2.00%	3.00%	0.00%	2.00%	1.00%	2.00%
12	0.00%	1.00%	1.00%	3.00%	3.00%	3.00%
20	0.00%	1.00%	2.00%	2.00%	4.00%	4.00%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	0.00%	2.00%	0.00%	3.00%	2.00%	0.00%
3	2.00%	1.00%	0.00%	1.00%	2.00%	0.00%
7	3.00%	2.00%	1.00%	3.00%	1.00%	7.00%
12	4.00%	0.00%	0.00%	0.00%	1.00%	3.00%
20	3.00%	0.00%	1.00%	1.00%	1.00%	2.00%

Figure 4.2: 10x10 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for the two best performing methods seen in Figure 4.1.

Figure 4.3 shows the quantization error for these methods using the best combination of parameters. The quantization error for the method using novel inputs tracks higher than the method without until close to the breaking point. This may have as much to do with the choice of σ as it does with the choice of method.

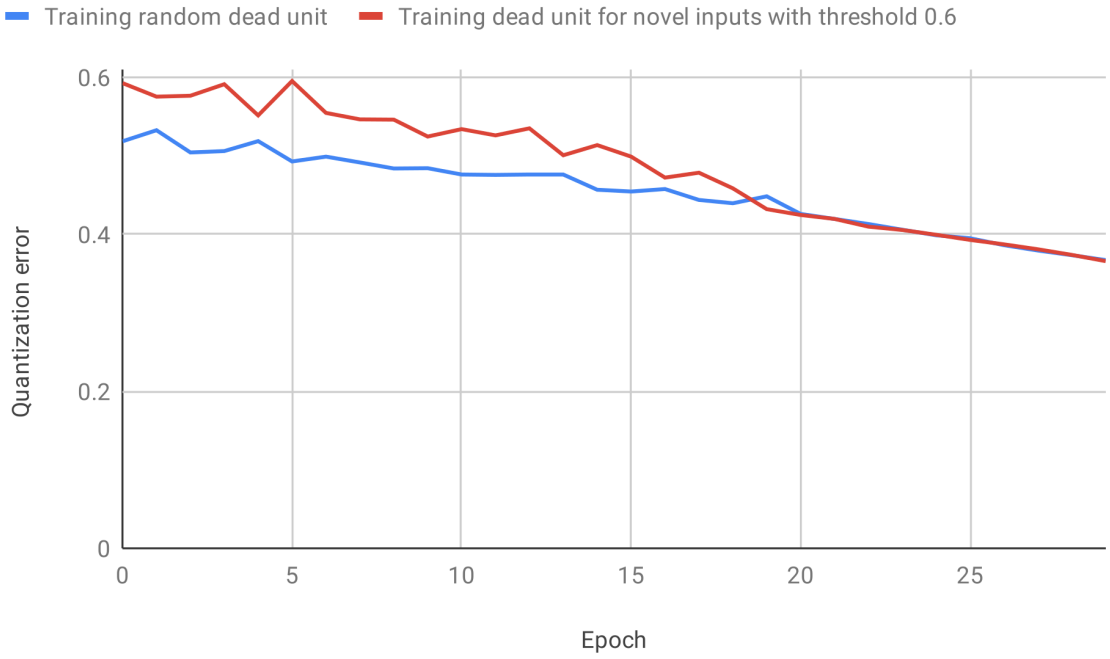


Figure 4.3: Comparison of quantization errors for two of the best performing parameter combinations. The blue line represents the quantization error for the **training random dead unit** method with initial $\sigma = 2.0$, breaking point at the 20^{th} epoch and α at the breaking point $= 0.5$. The red line represents the quantization error for the **training dead units for novel inputs with threshold 0.6** method with initial $\sigma = 5.0$, breaking point at the 20^{th} epoch and α at the breaking point $= 0.5$.

Figures 4.4 and 4.5 show the final map topology for the 10x10 grid for the two chosen methods / parameter combinations. The patterns which have been mapped to each neuron and which parts of the map would be most likely respond to a given input pattern can be seen here. One problem with smaller maps is that despite less dead units, each neuron present must represent more patterns. The chart bears this out with e.g. the 1 digit being spread across various parts of the map.

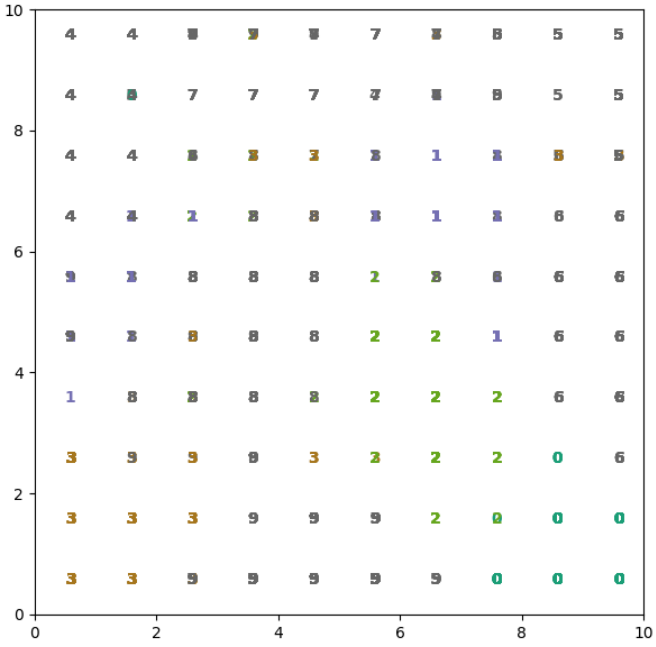


Figure 4.4: Winmap of the **training random dead unit** method.

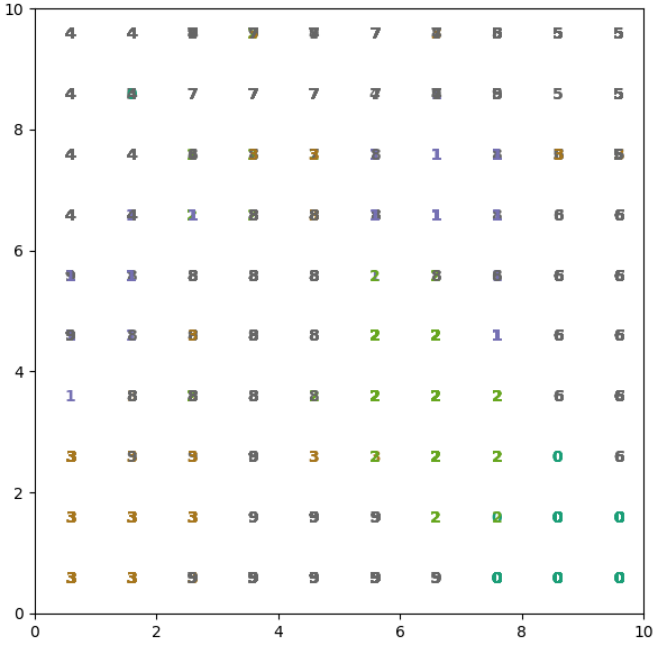


Figure 4.5: Winmap of the **training dead units for novel inputs with threshold 0.6** method.

4.2.2 20x20

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: [0%, 9%), [9%, 10%], (10%, 12%], (12%, 100%]. These results can be seen in figure 4.6.

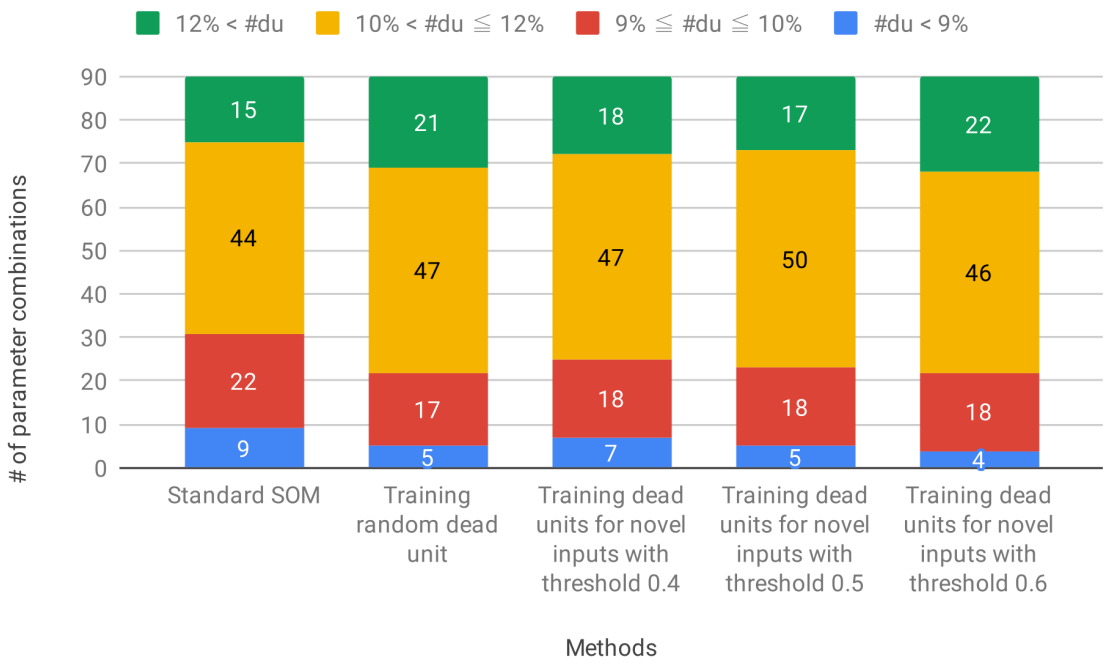


Figure 4.6: 20x20 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments. The chart similar to Figure 4.1. Here it is clearer that Standard SOM results in the most solutions with fewer dead units (#du).

Results indicate that the best performing method for threshold 10% is **standard SOM**. The performance of parameter combinations for this method can be seen in Figure 4.7.

Standard SOM

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	13.50%	11.50%	10.50%	11.75%	12.25%	10.50%
3	12.25%	10.75%	12.25%	12.75%	11.50%	11.75%
7	10.30%	11.30%	9.50%	10.00%	11.50%	12.80%
12	8.75%	8.25%	11.75%	9.75%	10.00%	10.25%
20	11.25%	10.50%	11.00%	10.50%	11.25%	11.25%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	13.00%	11.25%	9.50%	12.75%	12.00%	11.75%
3	10.50%	9.00%	13.50%	7.50%	13.25%	11.75%
7	10.50%	13.00%	7.00%	10.50%	11.30%	10.50%
12	12.25%	8.75%	9.50%	11.25%	7.75%	9.00%
20	10.00%	10.50%	9.75%	10.00%	10.75%	9.75%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	8.75%	9.50%	9.75%	11.25%	9.75%	9.25%
3	10.25%	8.50%	11.00%	9.25%	11.00%	11.25%
7	11.00%	12.00%	11.50%	12.50%	11.50%	10.50%
12	10.00%	8.50%	9.50%	10.00%	10.75%	10.00%
20	11.50%	12.00%	11.25%	9.75%	12.25%	12.50%

Figure 4.7: 20x20 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for standard SOM. It was seen in Figure 4.6 that this method performed best. The minimum percentage of dead units is found at initial sigma = 10, breaking point = 7 and alpha at breaking point = 0.2.

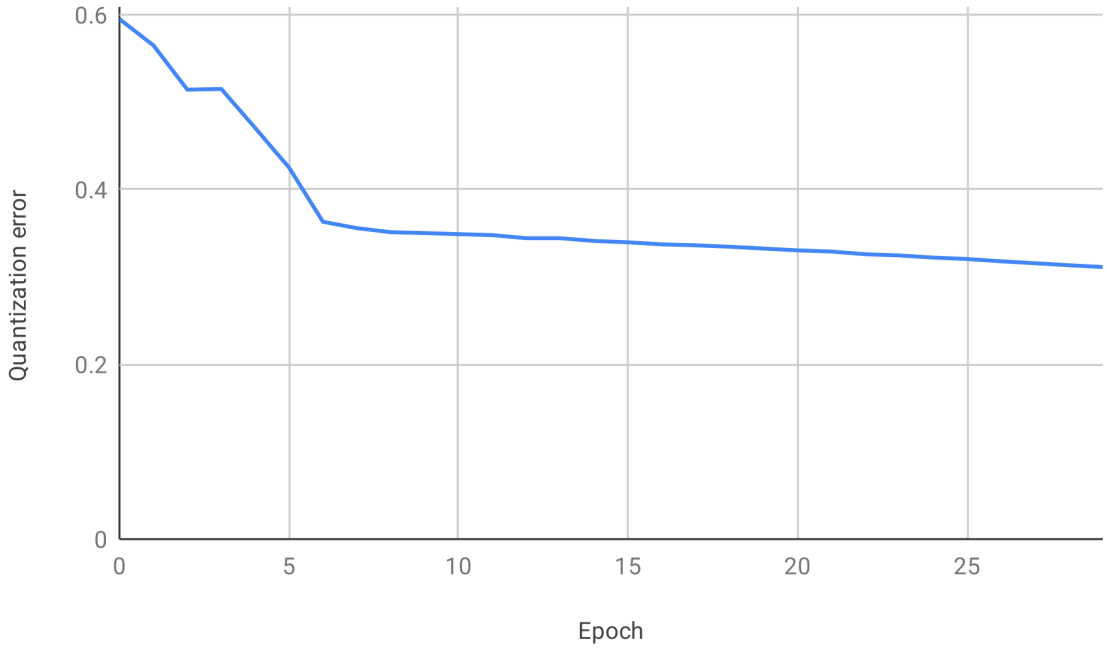


Figure 4.8: Quantization error of the best performing parameter combination on **standard SOM** with initial $\sigma = 10.0$, breaking point at the 7^{th} epoch and α at the breaking point = 0.2.

Figure 4.9 shows the map again, but this time for a 20x20 grid. The map looks to be organizing the groups a little better, but there are now islands of dead units which have appeared. These are perhaps due to there being large difference between the digits on either side of those islands. Note as discussed in Section 3.4, the digits 3 and 8 appear close to one another on the map.

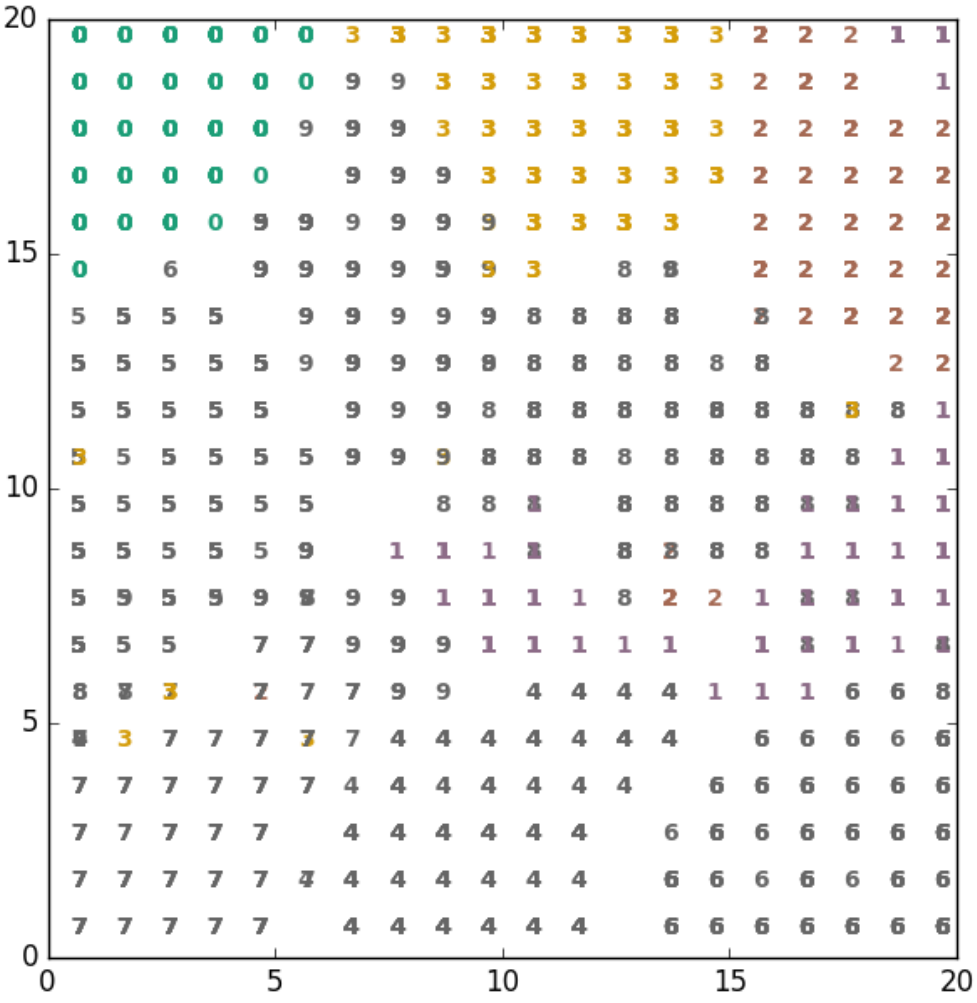


Figure 4.9: 20x20 MNIST: winmap of the best performing parameter combination.

4.2.3 40x40

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: [0%, 34%), [34%, 35%), [35%, 37%), [37%, 100%]. These results can be seen in figure 4.10.

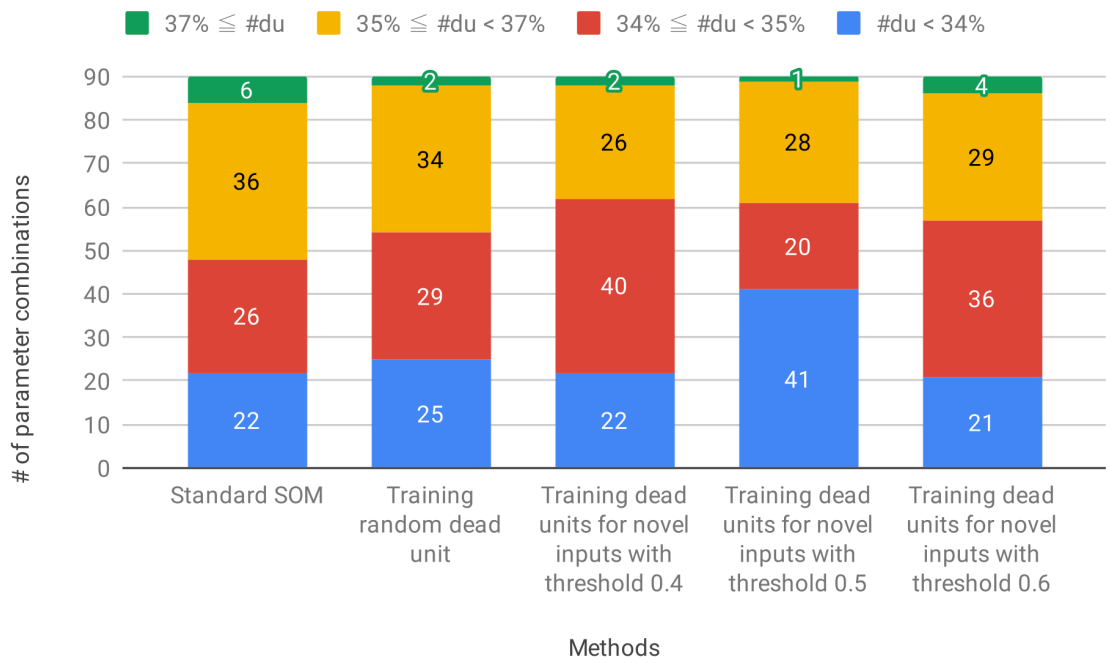


Figure 4.10: 40x40 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments. Again there is clear leader in terms of the number of parameter combinations leading to fewer dead units

Results indicate that the best performing method for threshold 34% is **training dead units for novel inputs with threshold 0.5**. The performance of parameter combinations for this method can be seen in Figure 4.11.

Training dead units for novel inputs with threshold 0.5

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	33.19%	33.69%	32.81%	33.50%	34.00%	33.88%
3	33.25%	33.38%	32.75%	34.75%	33.63%	33.19%
7	34.69%	34.06%	32.56%	32.88%	33.06%	33.50%
12	34.00%	33.44%	34.00%	32.38%	33.63%	35.31%
20	33.44%	35.44%	35.25%	33.81%	35.44%	35.31%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	35.13%	34.75%	35.25%	35.50%	34.19%	36.25%
3	34.25%	34.69%	35.50%	35.44%	34.56%	35.63%
7	35.69%	35.00%	33.31%	33.56%	35.06%	35.63%
12	33.69%	34.38%	36.19%	35.63%	35.13%	34.94%
20	35.63%	36.06%	34.19%	35.38%	36.31%	37.25%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	33.19%	33.69%	32.81%	33.50%	34.00%	33.88%
3	33.25%	33.38%	32.75%	34.75%	33.63%	33.19%
7	34.69%	34.06%	32.56%	32.88%	33.06%	33.50%
12	34.00%	33.44%	34.00%	32.38%	33.63%	35.31%
20	33.44%	35.44%	35.25%	33.81%	35.44%	35.31%

Figure 4.11: 40x40 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for the best performing method.

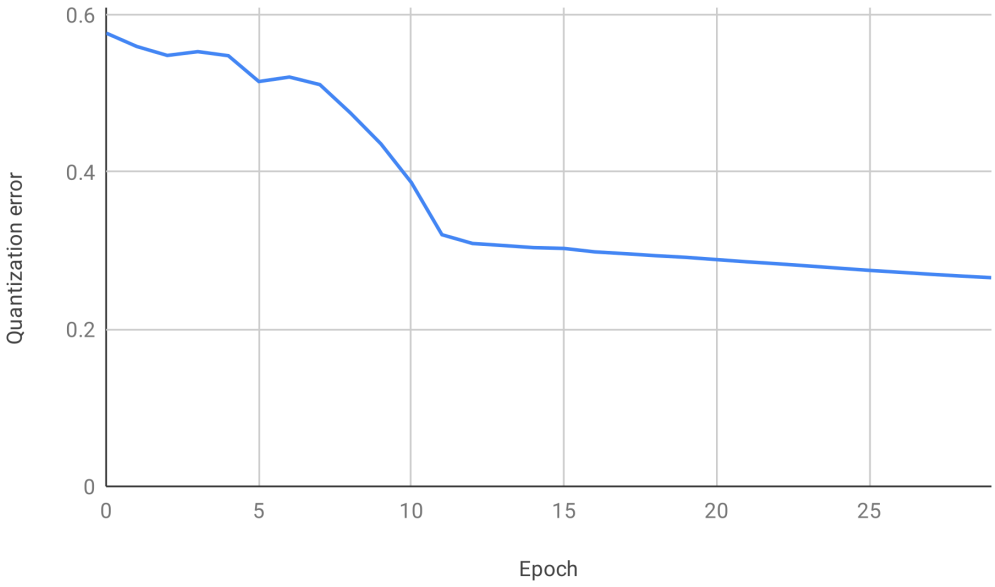


Figure 4.12: Quantization error of the best performing parameter combination on **training dead units for novel inputs with threshold 0.5** with initial $\sigma = 20.0$, breaking point at the 12^{th} epoch and α at the breaking point $= 0.5$.

The 40x40 map in 4.13 also includes some dead units as well as pockets of outliers. Overall however, there is a clear pattern on where each input would activate. The picture shows that, despite having approx. 32% of dead units, the map has a very decent organization.

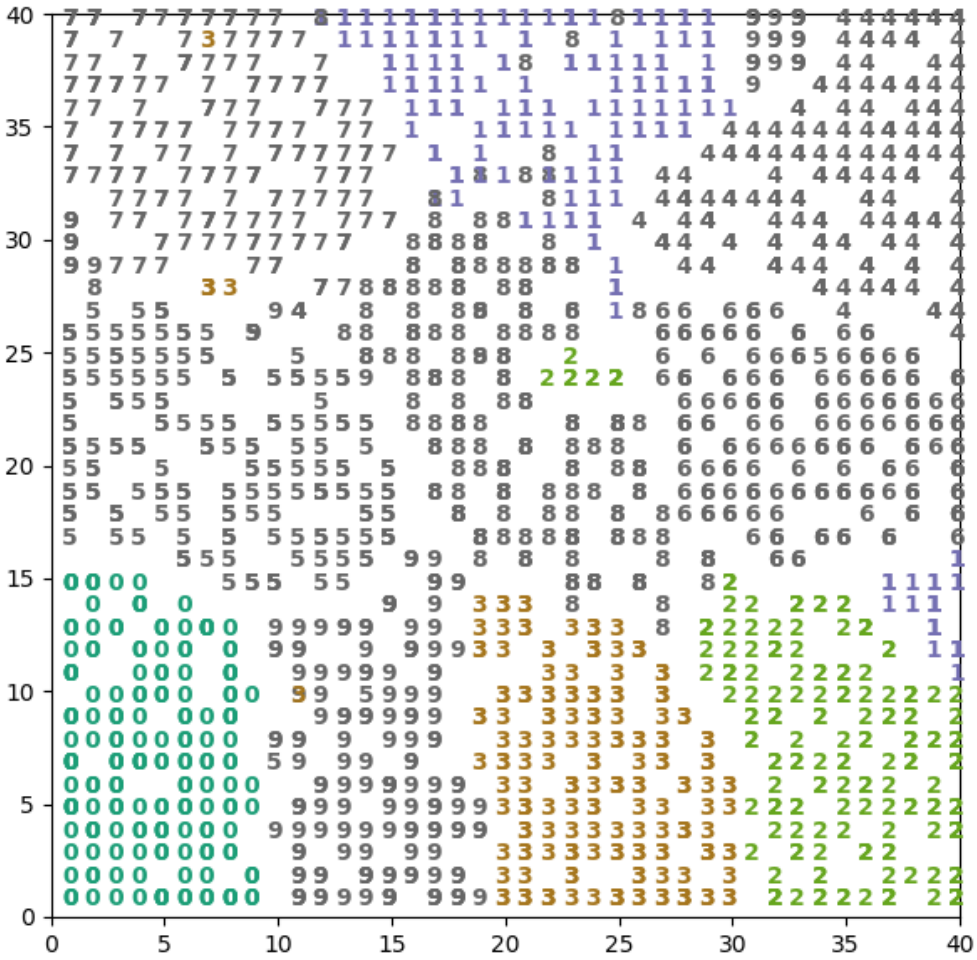


Figure 4.13: 40x40 MNIST - winmap of the best performing parameter combination.

4.3 RASTER

The raster dataset represents a low-dimensional input space (with no internal structure).

In this section, results from each of the map sizes trained - 10x10, 20x20 and 40x40 - will be discussed. Each subsection will take a closer look at the performance of individual methods on this specific data design and map sizes. It will also cover analysis of the training parameters for the methods which performed the best. The quantization error of the best performing methods and their best performing parameters will then be compared. All other results can be found in Appendix 5.2.5.

4.3.1 10x10

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: 0%, 1%, 2% and >2%. Ranges are not necessary here as a 10x10 map has exactly 100 neurons, so it is not possible to have e.g. 1.5% dead units. These results can be seen in figure 4.14.

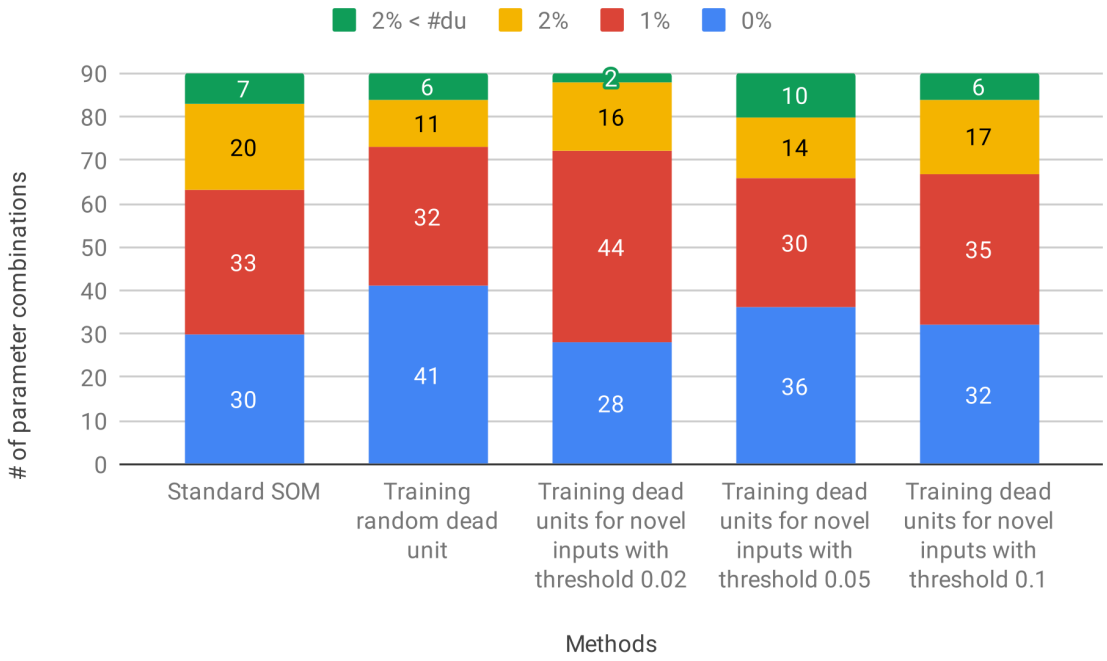


Figure 4.14: 10x10 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments.

Results indicate that the best performing methods for threshold 0% are **training random dead unit** and **training dead units for novel inputs with threshold**

0.05. The performance of parameter combinations for this method can be seen in Figure 4.15.

Training random dead unit

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	1.00%	1.00%	0.00%	1.00%	3.00%	1.00%
3	1.00%	0.00%	3.00%	0.00%	0.00%	3.00%
7	1.00%	1.00%	1.00%	0.00%	1.00%	0.00%
12	2.00%	4.00%	0.00%	1.00%	0.00%	0.00%
20	0.00%	2.00%	0.00%	0.00%	0.00%	0.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	0.00%	2.00%	1.00%	0.00%	1.00%	1.00%
3	0.00%	1.00%	1.00%	0.00%	1.00%	0.00%
7	0.00%	1.00%	1.00%	0.00%	3.00%	0.00%
12	0.00%	0.00%	1.00%	1.00%	1.00%	0.00%
20	0.00%	3.00%	0.00%	0.00%	0.00%	2.00%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	2.00%	0.00%	0.00%	1.00%	2.00%	1.00%
3	0.00%	0.00%	0.00%	0.00%	2.00%	0.00%
7	1.00%	1.00%	2.00%	1.00%	2.00%	2.00%
12	0.00%	1.00%	0.00%	1.00%	0.00%	1.00%
20	2.00%	1.00%	0.00%	1.00%	0.00%	1.00%

Training dead units for novel inputs with threshold 0.05

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	1.00%	1.00%	1.00%	0.00%	0.00%	1.00%
3	3.00%	0.00%	0.00%	1.00%	2.00%	1.00%
7	1.00%	0.00%	0.00%	2.00%	2.00%	1.00%
12	1.00%	1.00%	1.00%	0.00%	0.00%	2.00%
20	2.00%	2.00%	1.00%	1.00%	0.00%	3.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	2.00%	1.00%	0.00%	1.00%	0.00%	3.00%
3	0.00%	0.00%	1.00%	1.00%	0.00%	0.00%
7	1.00%	0.00%	0.00%	0.00%	1.00%	1.00%
12	0.00%	1.00%	0.00%	0.00%	3.00%	2.00%
20	1.00%	1.00%	0.00%	0.00%	1.00%	0.00%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	1.00%	1.00%	2.00%	0.00%	0.00%	0.00%
3	0.00%	3.00%	5.00%	2.00%	0.00%	0.00%
7	1.00%	1.00%	0.00%	3.00%	1.00%	2.00%
12	0.00%	0.00%	0.00%	2.00%	3.00%	0.00%
20	3.00%	1.00%	0.00%	2.00%	2.00%	7.00%

Figure 4.15: 10x10 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for the two best performing methods seen in Figure 4.14.

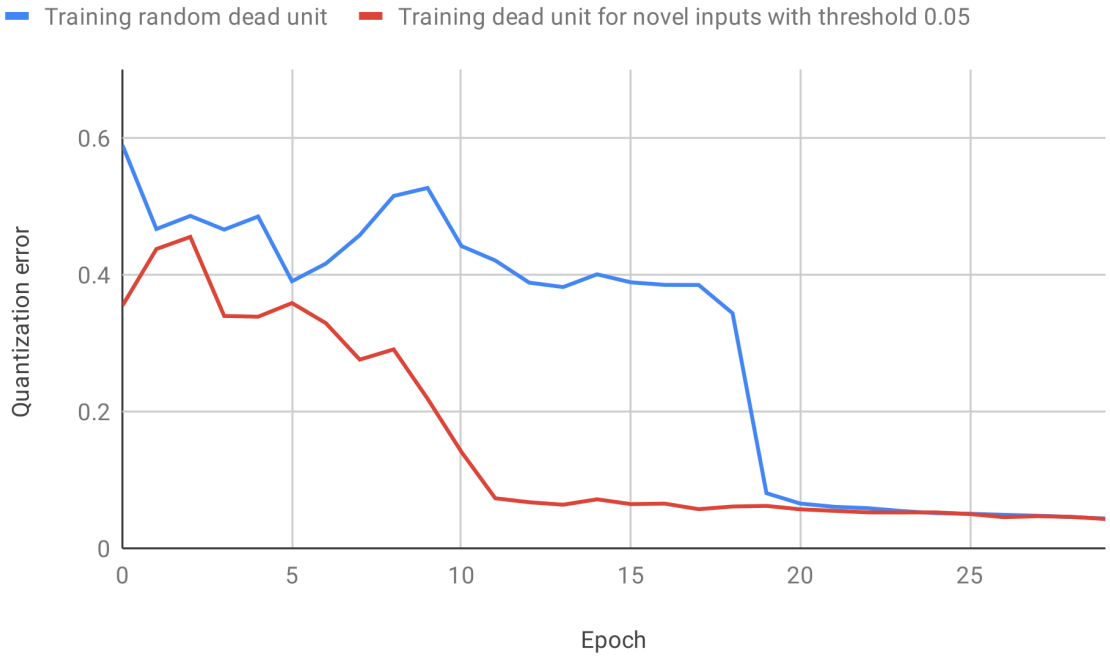
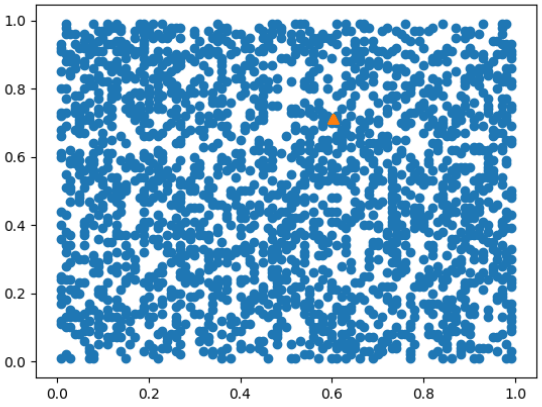
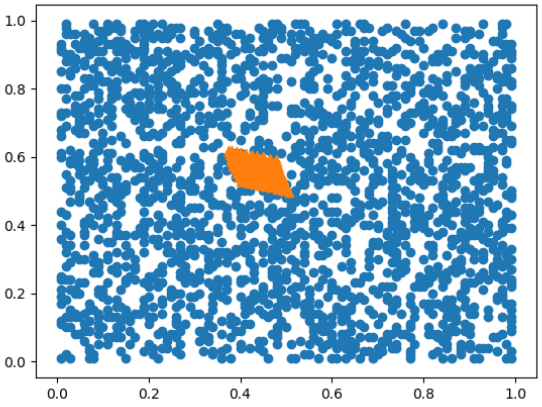


Figure 4.16: 10x10 map - comparison of quantization errors for two of the best performing parameter combinations. The blue line represents quantization error for **training random dead unit** method with initial $\sigma = 100.0$, breaking point at the 20^{th} epoch and α at the breaking point $= 0.1$. The red line represents quantization error for **training dead units for novel inputs with threshold 0.05** method with initial $\sigma = 10.0$, breaking point at the 12^{th} epoch and α at the breaking point $= 0.5$.

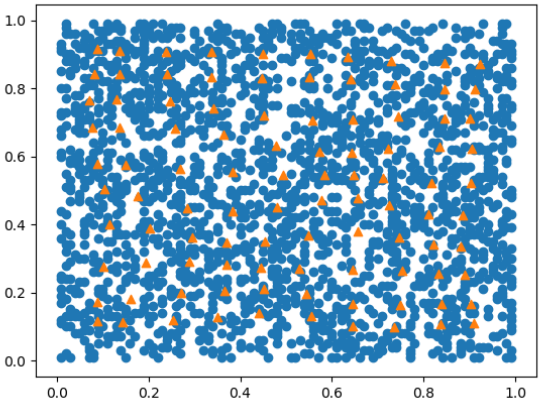
In Figure 4.16 both methods error converges to values around 0.04, meaning the maps are well trained.



(a) Neuron weights at the 1st epoch of **training random dead unit** method.



(b) Neuron weights at the 15th epoch of **training random dead unit** method.



(c) Neuron weights at the 30th epoch of **training random dead unit** method.

Figure 4.17: 10x10 map - visual representation of weights during training process. Blue dots represent the data, orange triangles represent neuron weights.

4.3.2 20x20

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: [0%, 9%), [9%, 10%], (10%, 12%], (12%, 100%]. These results can be seen in figure 4.18.

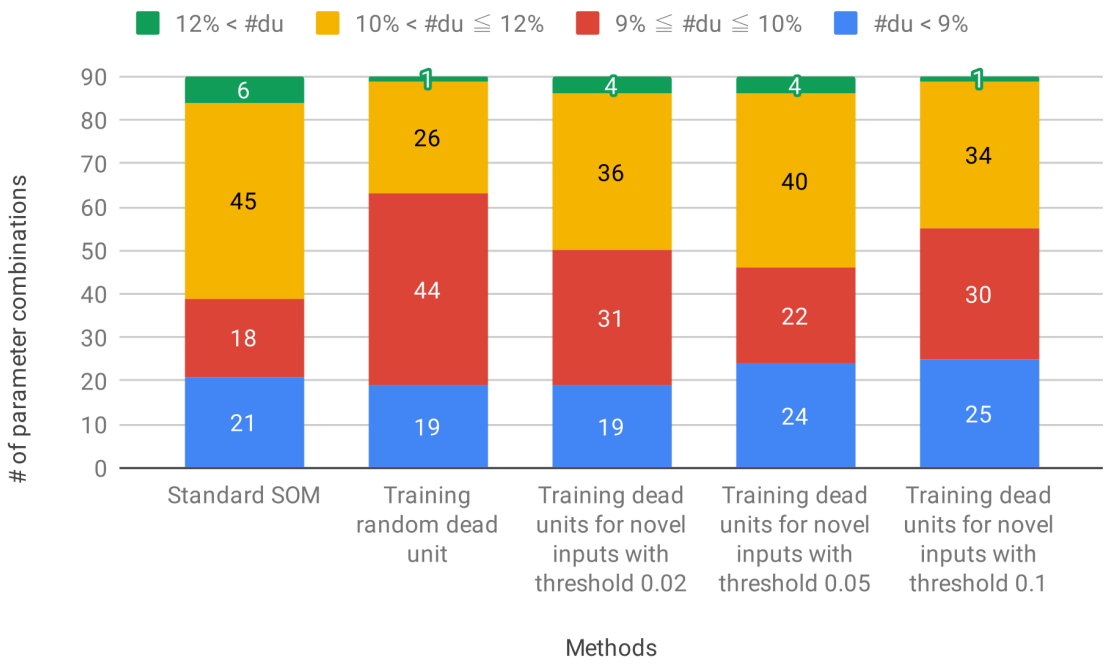


Figure 4.18: 20x20 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments.

Results indicate that the best performing methods for a threshold of 9% are **training dead units for novel inputs with threshold 0.05** and **training dead units for novel inputs with threshold 0.1**. The performance of parameter combinations for this method can be seen in Figure 4.19.

Training dead units for novel inputs with threshold 0.05

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	10.75%	8.00%	10.75%	10.50%	9.50%	11.00%
3	9.00%	10.00%	11.25%	12.00%	10.00%	9.00%
7	9.75%	11.25%	10.75%	7.75%	8.25%	10.75%
12	10.75%	10.75%	12.25%	7.50%	10.25%	11.75%
20	7.50%	10.50%	13.25%	10.50%	10.75%	11.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	8.00%	10.50%	11.00%	9.25%	7.75%	11.25%
3	7.75%	11.50%	9.75%	8.50%	9.25%	7.25%
7	10.00%	10.00%	12.00%	9.00%	8.75%	10.00%
12	10.25%	8.00%	8.50%	9.75%	10.50%	9.25%
20	10.50%	8.50%	10.00%	9.50%	12.50%	8.25%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	8.25%	10.25%	10.75%	11.00%	8.00%	11.00%
3	8.25%	11.00%	9.00%	11.75%	8.75%	9.25%
7	12.00%	7.75%	7.50%	11.25%	10.00%	10.50%
12	8.25%	8.50%	10.25%	9.75%	11.00%	5.50%
20	11.25%	12.50%	10.25%	9.00%	12.00%	10.25%

Training dead units for novel inputs with threshold 0.1

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	9.50%	10.50%	8.75%	8.50%	11.50%	12.00%
3	8.25%	9.25%	9.75%	9.25%	10.00%	9.25%
7	8.00%	7.50%	10.25%	8.75%	8.75%	10.50%
12	8.75%	9.75%	9.50%	7.25%	9.25%	10.25%
20	11.00%	9.75%	10.00%	11.00%	8.75%	10.50%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	9.75%	11.00%	11.00%	8.75%	10.00%	11.50%
3	11.50%	12.00%	8.75%	9.50%	9.75%	13.25%
7	10.00%	9.25%	9.00%	8.75%	9.25%	8.75%
12	9.75%	11.50%	11.00%	11.00%	10.50%	8.75%
20	9.00%	10.75%	10.00%	11.50%	9.25%	10.75%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	9.25%	10.25%	9.75%	10.50%	11.50%	8.25%
3	11.50%	8.75%	7.75%	10.00%	10.25%	11.00%
7	7.00%	11.75%	10.75%	8.75%	9.75%	8.75%
12	10.00%	11.25%	10.50%	8.00%	8.25%	10.25%
20	9.25%	7.25%	7.00%	10.25%	9.25%	11.00%

Figure 4.19: 20x20 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for the two best performing methods seen in Figure 4.18.

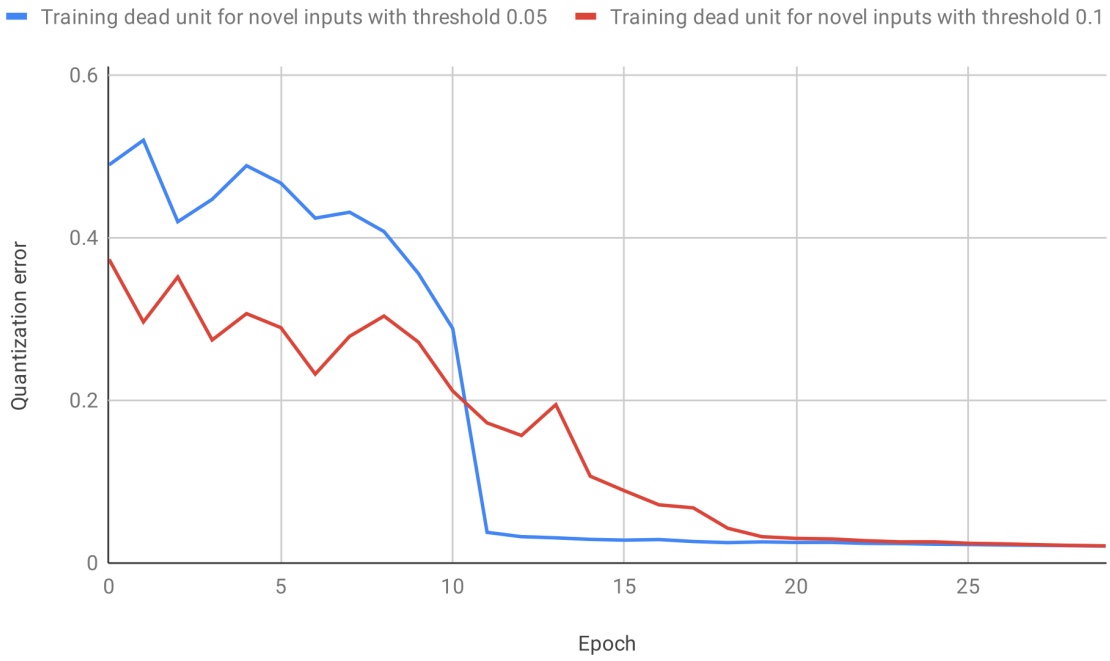
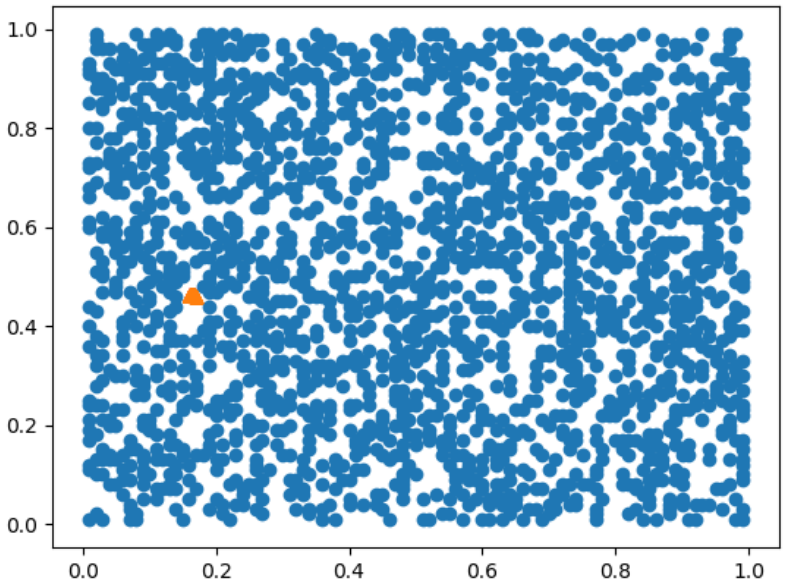
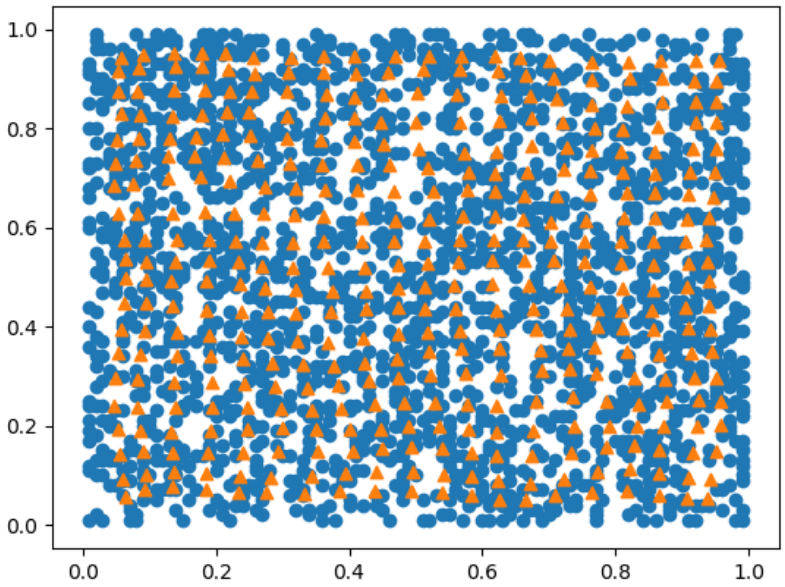


Figure 4.20: 20x20 map - comparison of quantization errors for two of the best performing parameter combinations. The blue line represents quantization error for **training dead units for novel inputs with threshold 0.05** method with initial $\sigma = 100.0$, breaking point at the 12^{th} epoch and α at the breaking point $= 0.5$. The red line represents quantization error for **training dead units for novel inputs with threshold 0.1** method with initial $\sigma = 10.0$, breaking point at the 20^{th} epoch and α at the breaking point $= 0.5$.

It is clearly visible in Figure 4.20 where breaking point for **training dead units for novel inputs with threshold 0.05** method is. Both maps are trained well, since they both converge to values around 0.02.



(a) Neuron weights at the 1st epoch of **training dead units for novel inputs with threshold 0.05** method.



(b) Neuron weights at the 15th epoch of **training dead units for novel inputs with threshold 0.05** method.

Figure 4.21: 20x20 map - visual representation of weights during training process. Blue dots represent the data, orange triangles represent neuron weights.

4.3.3 40x40

To compare methods on this data design and map size, an aggregation of the percentage of dead units into the following bins was chosen: [0%, 41%), [41%, 42%], (42%, 43%], (43%, 100%]. These results can be seen in figure 4.22.

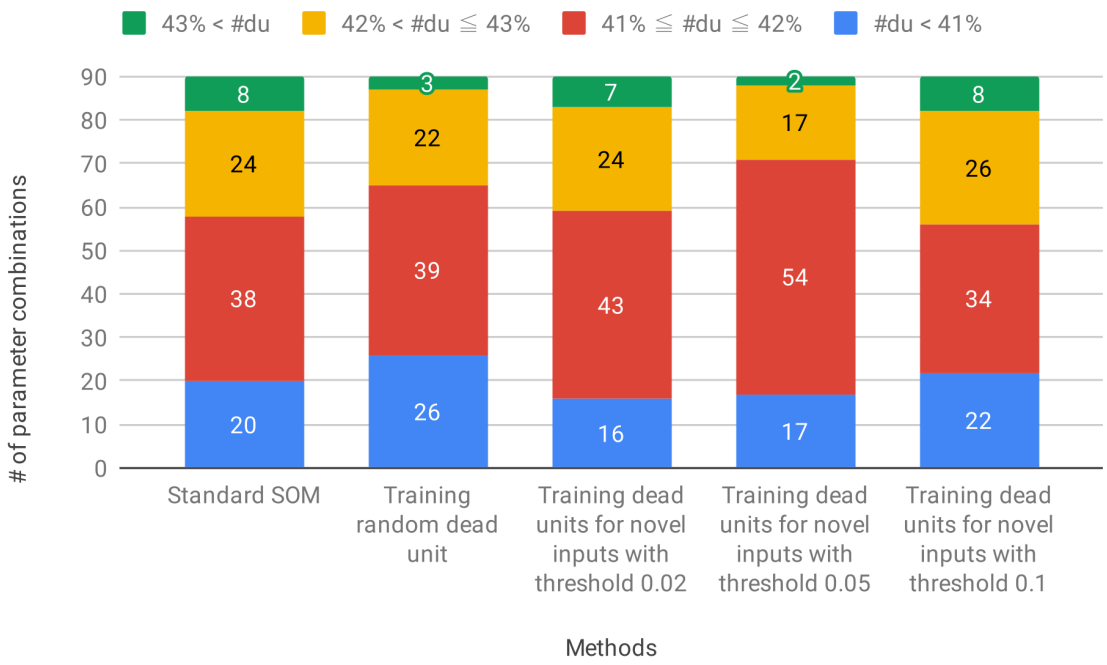


Figure 4.22: 40x40 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments.

Results indicate that the best performing methods for threshold 42% are **training random dead unit** and **training dead units for novel inputs with threshold 0.05**. The performance of parameter combinations for this method can be seen in Figure 4.23.

Training random dead unit

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	42.00%	42.31%	40.88%	40.88%	42.50%	42.63%
3	42.63%	39.88%	41.81%	41.56%	42.81%	43.00%
7	41.25%	41.06%	41.75%	41.63%	42.38%	42.69%
12	41.50%	41.63%	42.31%	42.63%	42.38%	43.25%
20	41.44%	41.81%	42.19%	41.25%	42.50%	43.56%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	43.69%	40.81%	42.13%	40.94%	41.19%	41.00%
3	41.69%	42.69%	41.81%	40.94%	42.31%	41.00%
7	41.38%	41.75%	41.25%	42.56%	41.94%	41.81%
12	42.06%	42.31%	41.44%	41.50%	41.25%	41.94%
20	40.94%	41.38%	40.94%	41.75%	41.75%	40.63%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	39.69%	39.75%	41.56%	40.81%	40.25%	40.31%
3	41.81%	40.19%	41.06%	40.00%	41.50%	39.81%
7	39.63%	41.75%	40.31%	40.75%	40.56%	40.25%
12	41.25%	42.19%	39.69%	41.00%	42.44%	40.38%
20	41.25%	41.25%	40.94%	39.25%	42.56%	41.31%

Training dead units for novel inputs with threshold 0.05

Breaking point alpha: 0.1	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	41.75%	41.44%	42.63%	43.81%	42.75%	42.25%
3	41.13%	40.69%	42.25%	41.13%	41.88%	41.81%
7	42.00%	41.50%	41.44%	42.56%	42.13%	42.88%
12	42.50%	41.63%	41.13%	40.69%	41.44%	41.69%
20	42.00%	42.56%	41.81%	41.94%	42.69%	43.00%

Breaking point alpha: 0.2	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	39.88%	40.38%	41.50%	40.69%	41.50%	41.69%
3	41.94%	41.50%	42.19%	41.63%	43.13%	41.88%
7	41.63%	41.63%	41.63%	41.94%	41.19%	41.69%
12	41.63%	41.69%	40.69%	42.19%	41.25%	41.44%
20	41.94%	41.31%	39.69%	41.06%	41.25%	41.75%

Breaking point alpha: 0.5	Initial sigma 2.0	5.0	10.0	20.0	50.0	100.0
Breaking point 1	42.44%	42.31%	41.63%	41.19%	41.44%	41.69%
3	40.56%	39.06%	42.00%	40.63%	41.56%	39.75%
7	42.06%	40.81%	41.06%	41.63%	40.94%	41.44%
12	42.25%	39.38%	41.50%	41.13%	40.88%	40.19%
20	41.63%	40.88%	41.69%	41.56%	41.63%	41.25%

Figure 4.23: 40x40 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point for the two best performing methods seen in Figure 4.22.

In Figure 4.24, quantization error of the best parameter combination from **training dead units for novel inputs with threshold 0.05** method has stabilized right at the start (see Figure 4.25). This could be caused by initial σ helping neuron weights organize well at the start and big adjustments in the topology were not needed. After 3rd epoch, σ drops to 1.0 and quantization error does not change much, since this is fine-tuning phase.

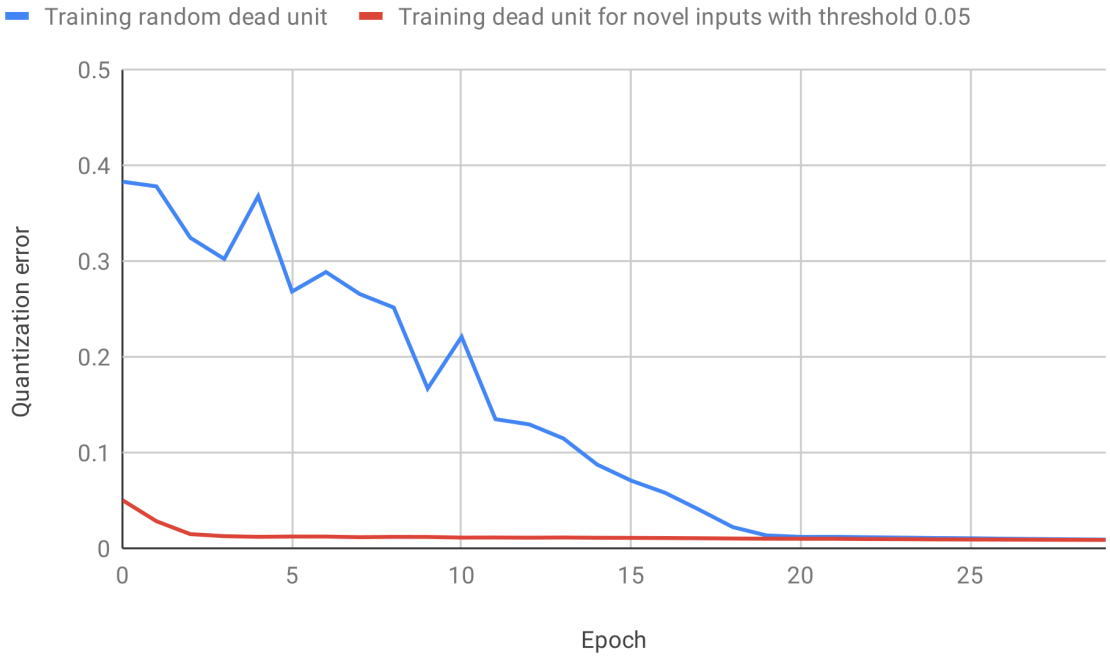
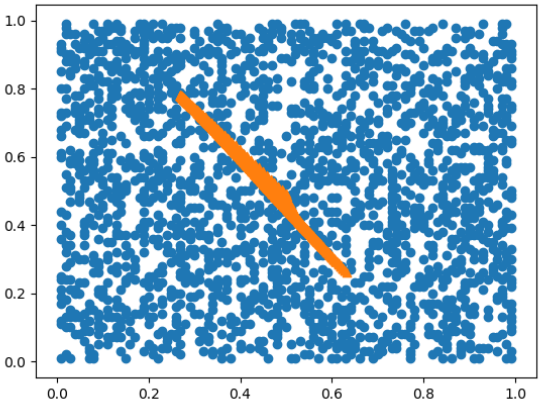
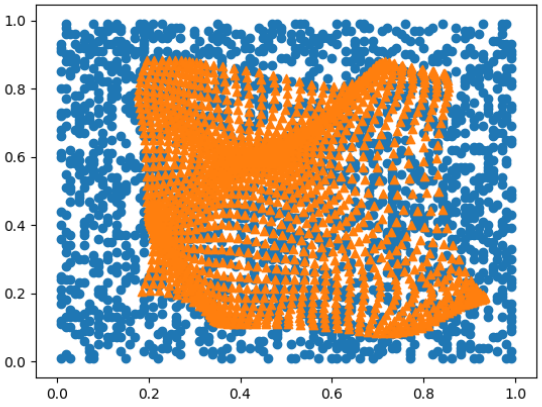


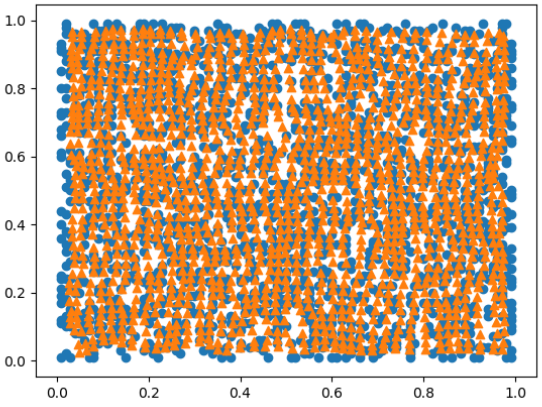
Figure 4.24: 40x40 map - comparison of quantization errors for two of the best performing parameter combinations. The blue line represents quantization error for **training random dead unit** method with initial $\sigma = 20.0$, breaking point at the 20^{th} epoch and α at the breaking point $= 0.5$. The red line represents quantization error for **training dead units for novel inputs with threshold 0.05** method with initial $\sigma = 5.0$, breaking point at the 3^{rd} epoch and α at the breaking point $= 0.5$.



(a) Neuron weights at the **1st** epoch of **training random dead unit** method.



(b) Neuron weights at the **1st** epoch of **training dead units for novel inputs with threshold 0.05** method.



(c) Neuron weights at the 30th epoch for both methods.

Figure 4.25: 40x40 map - visual representation of weights during training process. Blue dots represent the data, orange triangles represent neuron weights.

4.4 OMNIGLOT

As can be seen in Figure 4.26 this data design performed by far the worst for a 10x10 map, with no method / combination of parameters resulting in 0% dead units. It further performed much worse for both 20x20 and 40x40 maps. The later results can be seen in the Appendix 5.2.5.

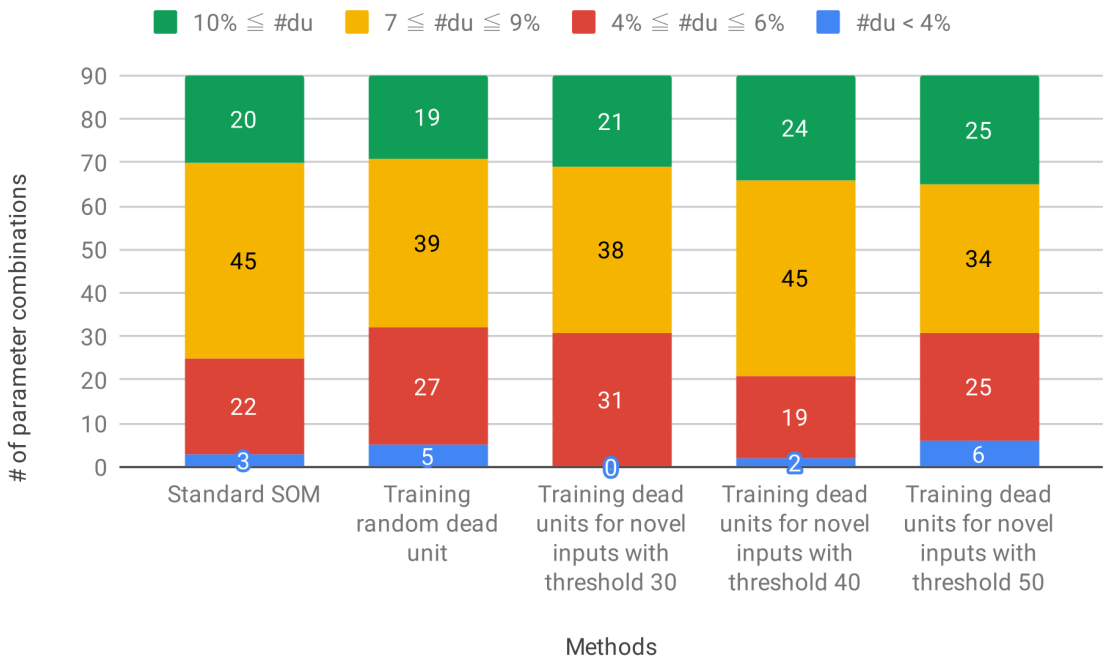


Figure 4.26: 10x10 map - performance of each method in terms of the percentage of dead units (#du) resulting from each of the 90 parameter combinations tested in the experiments.

The grid search of parameters (see Figure 4.27) is highly volatile suggesting small variations in the process of training the SOM are having a large effect.

Training dead units for novel inputs with threshold 50

Breaking point alpha: 0.1	Initial sigma					
Breaking point	2.0	5.0	10.0	20.0	50.0	100.0
1	5.00%	10.00%	3.00%	6.00%	9.00%	10.00%
3	7.00%	10.00%	13.00%	7.00%	6.00%	9.00%
7	9.00%	11.00%	3.00%	8.00%	7.00%	8.00%
12	10.00%	7.00%	16.00%	6.00%	6.00%	10.00%
20	7.00%	7.00%	13.00%	11.00%	6.00%	8.00%

Breaking point alpha: 0.2	Initial sigma					
Breaking point	2.0	5.0	10.0	20.0	50.0	100.0
1	6.00%	8.00%	7.00%	8.00%	13.00%	10.00%
3	11.00%	11.00%	5.00%	7.00%	2.00%	10.00%
7	6.00%	7.00%	5.00%	7.00%	10.00%	7.00%
12	10.00%	6.00%	6.00%	10.00%	7.00%	9.00%
20	4.00%	9.00%	10.00%	10.00%	8.00%	8.00%

Breaking point alpha: 0.5	Initial sigma					
Breaking point	2.0	5.0	10.0	20.0	50.0	100.0
1	5.00%	5.00%	7.00%	10.00%	10.00%	5.00%
3	9.00%	7.00%	7.00%	6.00%	8.00%	3.00%
7	8.00%	6.00%	5.00%	12.00%	6.00%	8.00%
12	5.00%	3.00%	8.00%	4.00%	5.00%	10.00%
20	9.00%	11.00%	8.00%	5.00%	3.00%	6.00%

Figure 4.27: 10x10 map - the percentage of dead units resulting from each combination of initial sigma, breaking point (epoch) and alpha at that breaking point.

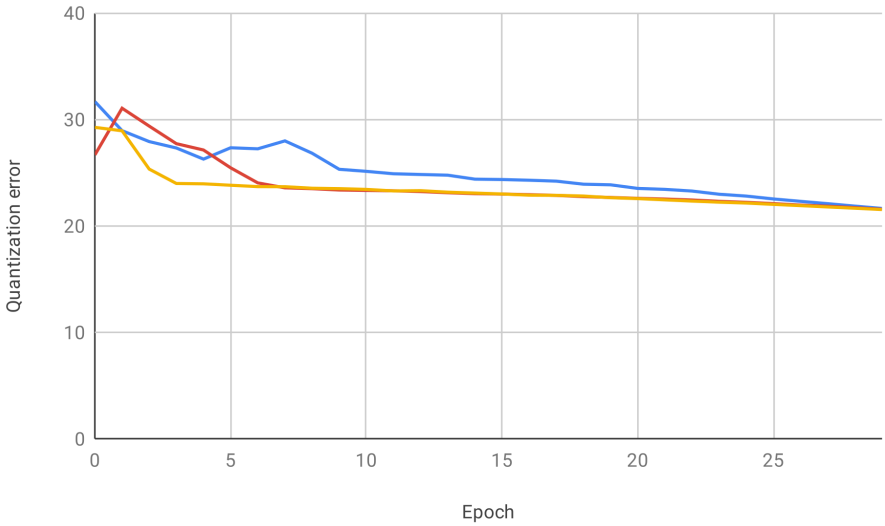


Figure 4.28: 10x10 map - comparison of quantization errors for three of the best performing parameter combinations. Each quantization error represents different parameter combination of **training dead units for novel inputs with threshold 50** method. The red line represents quantization error for combination with initial σ = 10.0, breaking point at the 7^{th} epoch and α at the breaking point = 0.1. The yellow line represents quantization error for combination with initial σ = 50.0, breaking point at the 3^{rd} epoch and α at the breaking point = 0.2. The blue line represents quantization error for combination with initial σ = 5.0, breaking point at the 12^{th} epoch and α at the breaking point = 0.12.

We conclude that that this type of multiclass data design is not suited for the methods that were tried. One method that might achieve better results is 1-shot learning.

Chapter 5

Discussion

The goal of this thesis was to create and experiment with different methods, trained on three different map sizes, of preventing dead units in self-organizing maps on the three different types of data designs (see Chapter 3.1).

First was **Neighbourhood size and learning rate annealing**, which is a standard approach to training SOM and we used it as a baseline.

Second was **Training a random dead unit**, where after each standard step, we randomly chose a dead unit, found the closest input sample and then adapted its weights and the weights of neurons within the winner neurons neighbourhood with this sample.

The third method, **Training dead units for novel inputs** we tested whether neurons weights were similar enough to the input sample, for which we created a threshold variable. We had hoped, this approach would let inputs of the same category to be clustered better together and novelty inputs would be mapped to unused neurons. Since we introduced a new threshold variable, we had to determine, what would be the correct value to distinguish inputs within the same category from the ones across categories. We calculated the mean Euclidean distance for inputs of the same category as well as the mean of cross-category Euclidean distances.

In this chapter the results will be discussed in order to recommend the best performing methods for each data design and future work will be presented.

5.1 Evaluation

Trivially, the smallest map size performed the best, since if one tries to reduce dead units, the smaller maps provide less space for dead units to occur. On smaller maps, the quantization error is often high, since several patterns are forced to map on the same neurons. One logical step to avoid this is to increase the map size. This does

not always help because some parts of the map can be left underutilized. At the same time, on smaller maps the neurons are already overcrowded with patterns, not leaving much room for dead units. There are therefore two competing facets to this problem:

1. A smaller map which has less dead neurons but higher quantization error.
2. A larger map which has lower quantization error but more dead neurons.

One way to deal with this problem is to use a larger map whilst trying other methods to reduce the number of dead units. The goal, then, is to find out whether the methods tested in this paper can utilize more of the potential of bigger map whist maintaining low quantization error.

To achieve the above, we experimented with 3 different methods, 3 different map sizes and 3 different data designs using many parameter combinations in a grid search.

5.1.1 Parameter space

The general conclusion for all methods was that a breaking point α closer to 0.5 usually provided the best results in terms of lower dead unit counts. A higher breaking point α means the learning rate will be higher for more epochs thus the neurons are able to move around more in the standard learning phase which may allow more dead units be adapted.

Another general conclusion is that extremely high values of initial σ (e.g. 50-100 and thus large initial neighborhood sizes) did not have the impact we expected in terms of the number of dead units. The hypothesis was that higher initial σ values would allow more outlying neurons to be pulled closer to the data. None of our results bore this out. One reason may be that this leads to high volatility with neurons being pulled violently (especially when the breaking point α is larger) in different directions at every step. Therefore the input data which is iterated through last would have drastically more impact on the map than inputs which were iterated through earlier in the process.

In terms of the breaking point no real pattern was found.

5.1.2 Data design

Training on the MNIST and RASTER datasets showed some promise. The final results for the OMNIGLOT dataset were comparatively disappointing. As previously mentioned, perhaps an alternative technique such as 1-shot learning would perform better on this type of data design (high number of classes with low number of samples per class).

5.1.3 Methods

Whilst there was a lot of overlap between methods (depending on the parameters chosen), the second method (**training random dead units**) and the third method (**training dead units for novel inputs** with either the mid - high range of the thresholds chosen) performed slightly better. For MNIST this meant thresholds of 0.5-0.6 for the third method, for RASTER it meant 0.05-0.1. As discussed earlier, results for OMNIGLOT were poor.

That said, no definitive conclusion can be drawn on which method is best without further experimentation. This will be covered in the next section.

5.2 Future work

In order to provide more concrete proof that the new methods discussed here can (or cannot) consistently prevent the occurrence of dead units, a number of further steps need to be taken. These can be broken down into 5 different categories.

5.2.1 More initialization of the neurons

Due to technological constraints (see Technology 5.2.4), it was not possible to run the experiments on more than one initialization per map. Trying out more of these would help ensure that number of dead units found for a particular parameter combination is close to the global minimum for that set.

5.2.2 Statistical tests

Since more runs were not possible, statistical tests could not be applied on the results.

5.2.3 Parameter space

The parameter space also had to be limited. A more granular exploration of the parameter space would be possible with more compute resources available. Combined with the above increase in initializations, this would be a powerful way to find the best method and parameter set for a given data design.

5.2.4 Technology

Technological constraints were discussed in Section 3.5. More powerful systems would enable more random starts and a more granular grid search on the parameter space.

5.2.5 Code

In addition to technology, some alterations to the code may be possible to parallelize some of the compute. In Section 3.6.3 this was discussed in detail, but overall this would enable us to make more out of the limits of our technology.

Bibliography

[1] E. Berglund and J. Sitte, "The parameter-less SOM algorithm," in *Proc. ANZIIS*, pp. 159–164, 2003.

[2] E. Berglund, "Improved PLSOM algorithm," *Applied Intelligence*, vol. 32, no. 1, pp. 122–130, 2010.

[3] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[4] D. J. Willshaw and C. Von Der Malsburg, "How patterned neural connections can be set up by self-organization," *Proc. R. Soc. Lond. B*, vol. 194, no. 1117, pp. 431–445, 1976.

[5] T. Kohonen, *Self-organization and associative memory*, vol. 8. Springer Science & Business Media, 2012.

[6] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[7] A. Baader and G. Hirzinger, "A self-organizing algorithm for multisensory surface reconstruction," in *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 1, pp. 81–88, IEEE, 1994.

[8] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[9] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[10] V. Kvasnicka, L. Benuskova, J. Pospichal, I. Farkas, P. Tino, and A. Kral, "Uvod do teorie neuronovych sieti," Iris, 1997.

[11] M. Strickert and B. Hammer, "Merge som for temporal data," *Neurocomputing*, vol. 64, pp. 39–71, 2005.

[12] B. Hammer, A. Micheli, A. Sperduti, and M. Strickert, "Recursive self-organizing network models," *Neural Networks*, vol. 17, no. 8, pp. 1061–1085, 2004.

[13] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[14] Kohonen, Teuvo, "Intro to SOM," 2005. `http://www.cis.hut.fi/projects/somtoolbox/theory/somalgorithm.shtml`.

[15] B. Lake, "brendenlake/omniglot," Feb 2019.

[16] T. Voegtlin, "Recursive self-organizing maps," *Neural Networks*, vol. 15, no. 8, pp. 979–991, 2002.

[17] M. Johnsson, C. Balkenius, and G. Hesslow, "Associative self-organizing map.," in *IJCCI*, pp. 363–370, 2009.

[18] D. DeSieno, "Adding a conscience to competitive learning," in *IEEE international conference on neural networks*, vol. 1, pp. 117–124, IEEE Piscataway, NJ, 1988.

[19] D. E. Rumelhart, J. L. McClelland, P. R. Group, *et al.*, *Parallel distributed processing*, vol. 1. IEEE, 1988.

[20] P. A. Estévez and R. Hernández, "Gamma som for temporal sequence processing," in *International Workshop on Self-Organizing Maps*, pp. 63–71, Springer, 2009.

Appendix A

Attached CD contains source code for all experiments as well as results.