INCA Open Examination Report

Exam Number: Y3603***

1 Discussion of architectures

The type of problem required to be solved with a neural network is a classification problem – that is, to decide class membership of an unknown data item, based on another data set of data items with known class memberships [1, Sec. 2]. Depending on the target output, the purpose could be to classify a set of inputs into two or more classes. This problem requires the simplest form of binary classification [2, Fig. 4], where the output should be either *yes* (room occupied) or *no* (unoccupied).

Given the nature of the problem, a wide range of feedforward architectures can be chosen. This section gives a brief discussion on the features of each architecture, and performs test-runs of that architecture's network in various configurations to find the best cursory performance. The MATLAB Neural Network Toolbox [3] will be used throughout this report, unless otherwise noted.

The data used in these cursory test-runs are directly imported from the data CSVs with minimal processing. The training dataset is used to train the networks, while the two test datasets merged into one (for now) is used to test the networks on unseen data. Input and target datasets are separated due to toolbox requirements. At this stage, the only pre-processing done is the removal of time, which will be later experimented with the chosen architecture.

1.1 Multilayer perceptron networks with backpropagation

Multilayer perceptron (MLP) is a typical feedforward neural network. In a MLP, neurons are arranged in layers, which consist of an input layer, one or more hidden layers, and an output layer. The feedforward property means that neurons are connected from one layer to the next with no 'lateral' or 'feedback' connections [4], and the backpropagation property through training functions implies the network's ability to propagate errors at the output layer back through the network to update weights during training, improving performance. MLP requires supervised training – with known target for the training dataset. MLP is useful to solve both regression and classification problems, but careful control of network configuration and training parameters is required to avoid overfitting. The selection of number of layers and layer sizes are important in creating accurate and useful MLP networks.

Three different backpropagation training functions are used for MLP: Levenberg-Marquardt (*trainlm*), gradient descent (*traingd*), and gradient descent with momentum (*traingdm*) for updating weight and bias values during training. All three are trained on the training dataset with various layer configurations to test for best performance. The performance here is measured by two indices: the mean square error (MSE) and the actual misclassification rate (with classification determined by rounding to 0 or 1), both on the testing dataset unseen during training, as in practical use the network nearly always works with unseen data. All other parameters are toolbox default. The results of the testing can be seen in Figure 1.

MLP Configuration	[5 3 2]	[6 4 2]	[10 5 5]	[20 10 10]	[6 4]
trainlm Validation MSE	0.077	0.090	0.062	0.093	0.045
trainlm Misclassification (%)	7.87	9.63	6.21	9.87	8.15
traingd Validation MSE	0.149	0.053	0.101	0.054	0.088
traingd Misclassification (%)	16.5	4.16	10.2	6.19	11.8
traingdm Validation MSE	0.046	0.197	0.109	0.033	0.032
traingdm Misclassification (%)	5.87	27.5	16.9	2.24	3.31

Figure 1: Results from cursory testing of multilayer perceptron (MLP) networks.

Barring the large [20 10 10] architecture (while the best external validation performance for *traingdm*, is slow to train and has a tendency to overfit other training data), the best performing architectures for *trainlm*, *traingd* and *traingdm* appear to be [10 5 5], [6 4 2] and [6 4] respectively. It is worth nothing that these are performed on data with limited preprocessing, and are indicative only to determine what configurations or similar configurations could be potentially used if MLP is chosen to be the final architecture.

1.2 Radial basis function networks

The radial basis function (RBF) network is another feedforward architecture with the ability to solve linearly inseparable classification problems. Rather than using hidden layers of identical neurons to achieve this like MLP, a RBF network transforms data into a higher dimension through the use of a layer of fully connected neurons computing radial basis functions [5]. This often allows the RBF network to solve classification problems more efficiently than MLP, but at the same time makes training on large data samples a slow process, due to the need of computing large numbers of distinct basis functions [6, p. 260].

The toolbox provides two methods for creating an RBF network: exact fit and fewer neurons. The former, while fast, is not used due to its high tendency of overfitting. The latter requires a definite goal MSE as stopping condition, thus requires much more time to train, but does not have the disadvantage of the exact fit method. Based on the MSEs seen during MLP training, four goals are used to train four different RBF networks, as shown in Figure 2. All other parameters are toolbox defaults.

RBF Training MSE Goal	0.2	0.1	0.05	0.01	0.001
Validation MSE	0.185	0.197	0.214	0.234	0.240
Misclassification (%)	24.33	24.31	24.31	24.28	24.27

Figure 2: Results from cursory testing of radial basis function (RBF) networks.

From past experience with the toolbox, training of RBF networks with the toolbox appears to be deterministic. It can be observed that RBF networks perform significantly worse than MLP networks (Figure 1) with the same input, especially in terms of actual misclassification rate on unseen data.

1.3 Self-organising maps

Self-organising maps (SOM) is a feedforward neural network architecture often used for specialised purposes such as data visualisation. The network consists of a mesh of connected neurons that attempts to rearrange positions match a data distribution in a iterative process [4, p. 34]. While it is often used to reduce the dimension of data for visualisation purposes, it can also be used to solve classification problems [7].

In particular, SOM is able to produce a out-dimensional output mesh from a multi-dimensional input dataset, allowing the computation of both linear regression and classification problems, as demonstrated by Haykin [6, Sec. 9.5]. This allows the binary classification problem to be solved by fitting a one-dimensional mesh to the training data, and determining which side of the mesh is a testing input placed. Further more, SOM deploys unsupervised learning and does not require targets for training data. This reduces the likelihood of bias towards patterns of training data, and allows continuous training in use.

The same training and testing datasets as used for MLP and RBF networks are used to evaluate SOM performance, with varying single-dimensional mesh sizes (also the number of neurons), as shown in Figure 3. Toolbox defaults such as *hexgrid* and *linkdist* are used.

SOM Mesh Size	[2 1]	[5 1]	[10 1]	[20 1]	[30 1]	[40 1]
Validation MSE	0.113	0.090	0.055	0.084	0.138	0.125
Misclassification (%)	24.3	11.3	7.33	3.21	4.83	3.74
SOM Mesh Size	[50 1]	[60 1]	[70 1]	[80 1]	[90 1]	[100 1]
Validation MSE	0.162	0.138	0.150	0.166	0.207	0.225
Misclassification (%)	11.1	10.8	9.89	11.7	30.2	29.7

Figure 3: Results from cursory testing of self-organising maps (SOM).

It can be observed that SOM exhibits a steady and accurate performance between 10 and 40 neurons in the mesh. The best performance with 20 neurons is comparable to those of MLP.

1.4 Selection of the final architecture

Based on the results from cursory testings of the three architectures discussed, as well their characteristics, the **self-organising map (SOM)** is chosen as the final architecture for further experimentation.

RBF is firstly ruled out due to its consistently high misclassification rate observed on unseen data, making it undesirable for the purpose of the problem: accurately identify the occupancy of a room based on sensor

data only. While different MLP training functions do produce comparable performance to SOM at different layer configurations, it is ultimately decided that the constant influx of new data during the operation of the occupancy identification system would create hassle in constantly sourcing data for the supervised training of MLP, as the patterns in sensor data will change as the season or other environmental conditions change. The unsupervised training of SOM allows the system to take into account new data in far less computational expensive way. Thus, MLP is also ruled out, leaving SOM as the chosen architecture.

In addition, a few more runs of the cursory testing also shows that the misclassification rates of MLP fluctuate more significantly than SOM, which could potentially cause the system to be less stable in producing accurate results if MLP is used in place of SOM.

2 Room occupancy detection with self-organising maps

2.1 Data exploration and pre-processing

While Section 1 merged the two testing datasets for convenience, according to the source of the data, the two testing datasets are different in nature: the much smaller first dataset was recorded mostly when the door to the room is closed, and the bigger second dataset was recorded mostly with the door open [2, Tb. 5]. Therefore, for the in-depth study of the data in this section, the two testing datasets will be processed separately. The number of inputs in the training dataset and the two test datasets (Validation 1, Validation 2) are 8143, 2665 and 9752 respectively.

Based on observations of results from *plotsomhits* in cursory testing, an assumption is made that the SOM will always class the data on either end of the one-dimensional network, which has always been the case in the observed results. This is used to calculate the misclassification rate, based on which end is closer to the location the data is placed by the SOM. When high misclassification rates are observed after preprocessing procedures later in this section, *plotsomhits* will be consulted again to ensure that the problem is not from this assumption. It is worth noting that the toolbox's SOM performs no preprocessing by default.

The training dataset will be used to train the SOM, while the two testing datasets will be used as unseen inputs to evaluate the performance of the trained SOM. There is no missing value. With the exception of time, all values are numerical. Therefore no imputation or numerical encodings are required. All three datasets exhibit a relative imbalance in class membership: the percentages of room unoccupied are 79%, 64% and 79% respectively. While is could potentially cause a bias in training and testing data, it was decided that no rebalancing measures should be conducted, as (1) the training and testing datasets exhibit a common pattern, and (2) this appears to be the normal usage pattern of the room in question, as well as the fact that 79% is not an overwhelming majority that could skew results, such as those observed in using SOM to detect transaction fraud (negative >99%) [8], which would require special measures for the SOM to function reliably.

To further illustrate the effects of rebalancing, a comparison test was conducted with toolbox defaults between a training dataset of 1729 randomly sampled unoccupied inputs and all 1729 occupied inputs, and a training dataset of 3458 randomly sampled inputs with no particular pattern. Each side of comparison was freshly trained ten times, and the results are as shown in Figure 4, demonstrating that it is better *not* to rebalance the inputs in SOM training. It is worth noting that as SOM deploys unsupervised training, the network is not aware of the class distribution of the training dataset.

Rebalance of Input Dataset	Rebalanced (1729 Positive + 1729 Negative)	Not Rebalanced (3458 Random)
Mean Training MSE	0.096	0.055
Mean Validation 1 MSE	0.071	0.069
Mean Misclassification 1 (%)	8.96	2.18
Mean Validation 2 MSE	0.112	0.140
Mean Misclassification 2 (%)	16.08	9.23

Figure 4: Results from testing the balancing of training input on a [20 1] SOM. (All MSE variances < 0.01, all % variances < 0.02.)

Two types of data based on time are provided: the sequence number and the exact time. Both of which could be potentially useful for the network, especially in studying the time-based occupation pattern of the room studied. But at the same time, this could cause the loss of generalisability, as well as affecting the network performance. In order to determine the suitability of using time, the best SOM configuration Section 1.3 – [20 1] is trained by three variations of the training dataset: time removed, with time sequence numbers, and with exact times

converted into integer timestamps. In the latter two variations, the time variable is scaled to between 0 and 1. For all variations, the other five input variables are attached as is. The results of the testing can be found in Figure 5.

Time Variation in Input	No Time	Integer Timestamps	Time Sequence Numbers
Validation 1 MSE	0.063	0.063	0.062
Misclassification 1 (%)	2.18	2.18	2.18
Validation 2 MSE	0.090	0.089	0.090
Misclassification 2 (%)	3.51	3.41	3.48

Figure 5: Results from testing the use of time in training input on a [20 1] SOM.

From the results, it can be seen that the inclusion of time (in either form) has almost no effect on the performance of network on unseen test data, confirming the assessment specification that the other input variables (from sensor data) have already included the influence of time. Therefore, for the consideration of generalisability, time is removed from the input variables.

This leaves five input variables, all based on sensor data, as shown in Figure 6.

#	Input Variable	Type	Range
1	Temperature	Numeric	[19, 24.4083]
2	Humidity	Numeric	[16.745, 39.5]
3	Light	Numeric	[0, 1697.2]
4	CO_2	Numeric	[412.75, 2076.5]
5	Humidity Ratio	Numeric	[0.0027, 0.0065]

Figure 6: Input variables to the SOM.

With five numeric input variables, two different forms of further pre-processing are possible: [0, 1] normalisation or principle component analysis (PCA) after variance standardisation. In order to better understand the characteristics of the input data, *plotmatrix* is first performed on input variables in Figure 6, as shown on the left side of Figure 7.

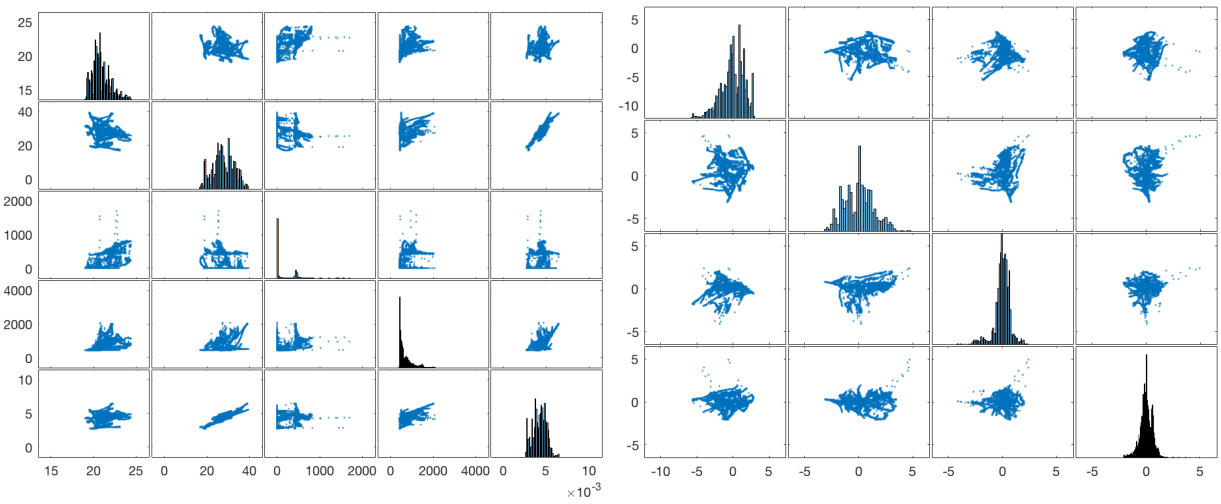

Figure 7: Results of *plotmatrix* of the full dataset before (left) and after (right) PCA.

From Figure 7(left), it is clear that some input variables may be skewed, especially #3 (Light) and #4 (CO_2), which may also contain outliers. This makes [0, 1] normalisation with *mapminmax* potentially unsuitable, as proved by comparison test with unnormalised data, shown in Figure 8.

An alternative to normalisation is to first transform input data so that each variable has a mean of 0 and standard deviation of 1, by first applying *mapstd* then *processpca* with a maximum fraction of 0.02. This resulted in the removal of one variable – which makes sense, as variable #5 Humidity Ratio was calculated from temperature and relative humidity, resulting in a possible correlation. The result of PCA is visualised in Figure 7 (right). It is evident that the processed input variables are far less correlated, and could potentially improve the training of SOM.

Normalisation of Input Dataset	Normalized	Not Normalized
Mean Training MSE	0.198	0.050
Mean Validation 1 MSE	0.138	0.064
Mean Misclassification 1 (%)	16.03	2.18
Mean Validation 2 MSE	0.320	0.101
Mean Misclassification 2 (%)	60.53	4.58

Figure 8: Results from testing the [0, 1] normalisation of training and testing inputs on a [20 1] SOM. (All MSE variances < 0.01, unnormalised misclassification 2 variance $= 5.74$, rest < 0.01.)

However, it came to a surprise that this is not the case, after a comparison testing of 10 runs with toolbox defaults on both a [20 1] and a [20 5] network, as shown in Figure 9.

PCA Processing of Input Dataset	After PCA	Before PCA
Mean Training MSE	0.467	0.050
Mean Validation 1 MSE	0.552	0.063
Mean Misclassification 1 (%)	84.65	2.17
Mean Validation 2 MSE	0.303	0.100
Mean Misclassification 2 (%)	38.70	4.55

Figure 9: Results from testing the PCA processing of training and testing inputs on a [20 1] SOM. (All MSE variances < 0.01, unnormalised misclassification 2 variance $= 5.50$, rest < 0.22.)

In order to confirm that the high misclassification rates do not originate from a problem in the assumption of SOM clustering on two ends, and that using a two-dimensional SOM instead will not alleviate the issue, SOM sample hit plots from the training of the [20 1] and another [20 5] SOM are shown in Figure 10. It can be observed that in both cases, more than two clusters of the data can be seen for each class, rendering a meaningful binary classification impossible.

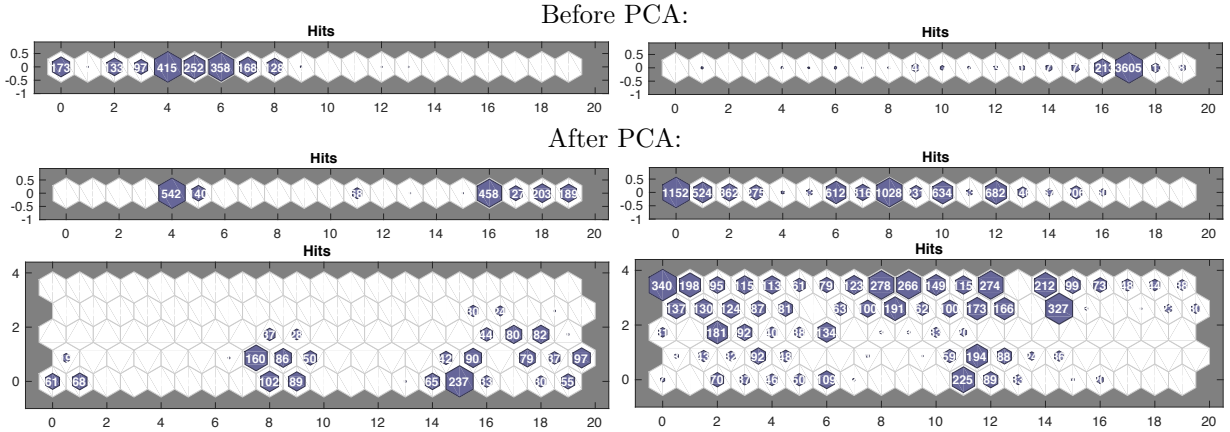

Figure 10: *plotsomhits* plots from the training of a [20 1] and a [20 5] SOM on the input data. At the top is the control group (before PCA processing), and at the bottom are SOM plots after PCA (with new SOMs). To the left are occupied cases, and to the right are unoccupied cases.

A possible explanation to this interesting observation is that SOM can be considered as a non-linear generalisation of PCA itself [9, p. 1]. Principle component could be used as a method for initialisation of the SOM weight positions, but it is evaluated to perform less well than random initialisation. Therefore, applying PCA on the inputs of SOM may not achieve the same effects as on the inputs of a MLP.

Considering the fact that the five variables shown in Figure 6 without additional pre-processing already performs very well (single-digit misclassification percentages) as the baseline for the few comparisons performed in this section, it was decided that **the five input variables will be used as is for the SOM without further processing**.

2.2 Description of the SOM architecture

The self-organising map (SOM) represents a unique feedforward architecture. The architecture consists of a mesh of connected neurons, each selectively tuned for specific features of the input data during the training process [6, Sec. 9.1]. The mesh of neurons are topologically ordered [4, p. 11], which implies that a specific point on the map formed by these neurons maps to a specific set of characteristics seen in a certain set of input data. This feature allows SOM to be used to solve simple classification problems such as this one, in addition to its very useful purposes in projecting high-dimensionally data nonlinearly to achieve purposes such as visualisation. Two models of this mapping process exist: the Willshaw-von der Malsburg's model and the Kohonen model [6, Sec. 9.2], and the MATLAB toolbox uses the Kohonen model [10], to be described in more detail in Section 2.3.

The SOM is implemented with the *selforgmap*/*newsom* and associated methods in the MATLAB Neural Network Toolbox [3]. For all of the preliminary testings in the previous sections, the default parameters for the network are used: 100 epochs in each phase of the training (phases to be covered in 2.3), initial neighbourhood size of 3 (to be later reduced), arranged in a hexagonal topology with linked distance as the distance function. In addition, the legacy *newsom* method also allows the creation of same SOMs with adjustable learning rates, at this time they are set to defaults. Later in Section 2.3, changes to some of these parameters will be explored.

The architecture of the SOM is relatively simple to implement: an input layer of a set number of neurons connects to a single layer of map neurons (nodes) topographically arranged in a mesh, the dimensions of which are defined by the network configuration. This forms a feedforward structure [6, p. 429]. Each map neuron has a set of weights in its connections with neurons on the input layer, which affects the length of the connection, resulting in the surface of the mesh to "bend" as the weights change. At initialisation, various methods can be used to initialise these weights. The toolbox default creates linearly separated sets of weight, and places the same number of neurons on the input layer as there are input variables, as shown in Figure 11.

Lateral connections are also to be created between neighbouring neurons based on the topology. Their positions are useful to the distance function such as linked distance (which means distances are immutable in this case, as *linkdist* counts by neuron only), as discussed above. In the case of the one-dimensional mesh used in this report, the lateral connections are much simpler due to the simplified structure (at most two connections per neuron), as shown in Figure 11 (left). A visual illustration of another SOM with dimensions [5 2] is shown in Figure 12.

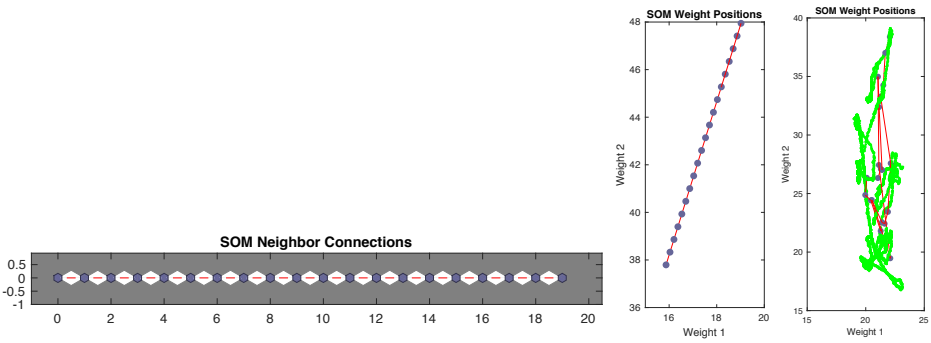

Figure 11: The initial neighbour connections and weight positions (first two of five) of the SOM created by the toolbox (left), as well as the weight positions of the same inputs after training (right).

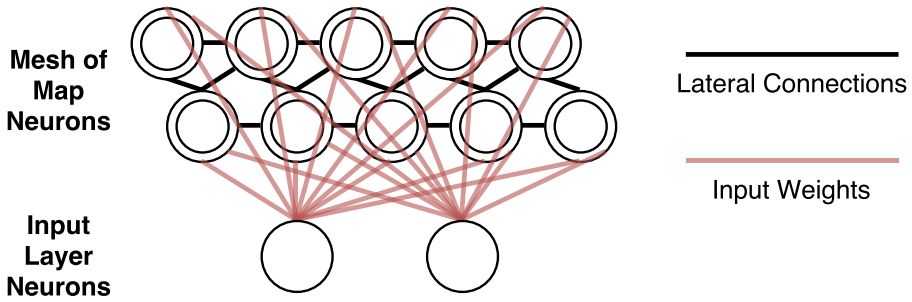

Figure 12: Visual illustration of a SOM with dimensions [5 2] and two input neurons (possibly connecting two input variables). The mesh is arranged in *hextop* grid.

2.3 Training of the SOM architecture

Unlike MLP, a Kohonen SOM has a defined way of training, implemented in the toolbox as *trainbu*. The training is divided into two, usually equal-length phases: the ordering phase and the convergence phase [4, p. 23]. The purpose of the training process is to expose all neurons to a sufficient number of different input patterns with different corresponding active spots on the map [6, p. 429], adjusting the input weights and therefore allowing the network to produce distinct responses for different classes of input patterns. Haykin [6, p. 435] recommends 1000 steps for each phase of the training. The MATLAB toolbox default is 100 for each phase [10], which produced a general classification accuracy of $> 95\%$ in cursory and data testing as shown in the previous sections.

The two training phases are procedurally identical, but different in the values of learning rate (η) and neighbourhood size (σ), both of which smaller in the convergency phase than the ordering phase, as the ordering phases is responsible for main topological ordering, while the convergency phase is used for fine tuning between neighbouring neurons [4, p. 23]. The starting learning rates are $\eta = 0.9$ and $\eta = 0.02$, and the start neighbourhood sizes are the entire mesh's radius and $\sigma = 1$ respectively for the ordering and convergence phases [11]. These values are close to the recommended start values [4, p. 23]. During each phase, exponential functions gradually reduces these parameters, as described below.

Each step during either phase is a three-stage process: competition, cooperation and adaptation [6, pp. 429-430]. The first stage determines the most excited (winning) neuron, which is used to determine a neighbourhood in the second stage, through a neighbourhood function h. Neurons in the neighbourhood selected then have their weights adjusted to be closer to the most excited neuron in the last stage. This process repeats until a defined number of steps are completed for each phase.

In the competition phase, with an input space of m variables and the particular input x, the selection of the most excited neuron $i(x)$ on a mesh of neurons (each neuron j with a weight vector w_j) is with the following algorithm [6, p. 430]:

$$i(x) = arg\min_j \|x - w_j\| \text{ , where } x = [x_1, x_2, ..., x_m]^T \text{ and } w_j = [w_{j1}, w_{j2}, ..., w_{jm}]^T \tag{1}$$

In the cooperation phase, owing to the observation that neurons closer to the winning neuron assimilate more towards the winning neuron than the neurons further away, the neighbourhood function h determines the scale of activation, based on the set neighbourhood size σ, and the distance d_{ji} from the winning neuron determined by a distance function, such as linked distance (*linkdist*) [6, p. 431]:

$$h_{ji} = exp(-\frac{d_{ij}^2}{2\sigma^2}) \text{ for lateral distance between the winning neuron } i \text{ and neighbouring neuron } j \tag{2}$$

And finally, in the adaptation phase, neurons in the neighbourhood of the winning neuron will assimilate towards the winning neuron. The new weight $w_j(n+1)$ of neuron j is updated at the $n+1$ step as followed [4, p. 21]:

$$w_j(n+1) = w_j(n) + \eta h_{ji}(n)(x - w_j(n)) \tag{3}$$

To achieve a gradual process of weight adjustment, the learning rate η and the neighbourhood size σ are generally exponentially reduced over time [4, pp. 21-22]:

$$\eta(n) = \eta_0 exp(-\frac{n}{\iota}) \qquad \sigma(n) = \sigma_0 exp(-\frac{n}{\iota'}) \tag{4}$$

While there is no choice of training algorithms, it is possible to alter the number of steps in each phrase of training, as well as the initial learning rates η and neighbourhood sizes σ to observe their effects on the performance of the trained SOM. Based on the recommendations from the module [4] and Haykin [6], several different combinations of learning rates and neighbourhood sizes are chosen, each tested ten times on a fresh [20 1] SOM in comparison with the baseline, as shown in Figure **??**.

2.4 Evaluation of the trained SOM

Before starting the evaluation of the trained SOM, it is necessary to first discuss in more detail how the actual misclassification rates ("Misclassification %") are calculated. As described at the start of Section 2.1, an assumption was made that the one-dimensional SOM will cluster the two classes of inputs at the two ends of

the map. During the course of this report, extensive observations on *plotsomhits* have always showed that this is the case on one-dimensional SOMs without normalisation or PCA conducted on the input data.

It was also observed that the positions of the two clusters have always been opposite to the ordering of the output vector (occupied/'1' inputs cluster on the left, unoccupied/'0' inputs cluster on the right. This is however, a projection of data in higher dimensions, and maybe arbitrary [12, p. 10]. While it appears that the toolbox implementation always projects the output data in this orientation, this is not a defined feature of the network.

Therefore, it was possible to infer the class of an input, based on its position in the output vector from applying the SOM on that input. However, as the assumptions made may be implementation specific (i.e. to the MATLAB toolbox), some manual observations on the results of other implementations may be required before these assumptions can be carried over.

References

[1] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of Biomedical Informatics*, vol. 35, no. 5–6, pp. 352 – 359, 2002.

[2] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and co 2 measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28–39, 2016.

[3] T. Kohonen, "Matlab implementations and applications of the self-organizing map," *Unigrafia Oy, Helsinki*, 2014.

[4] S. O'Keefe, "Self-organisation," Lecture notes for INCA, Department of Computer Science, University of York, November 2016.

[5] ——, "Radial basis function networks," Lecture notes for INCA, Department of Computer Science, University of York, November 2016.

[6] S. Haykin, *Neural Networks and Learning Machines: A Comprehensive Foundation*, 3rd ed. Pearson, November 2008.

[7] J. Owens and A. Hunter, "Application of the self-organising map to trajectory classification," in *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on.* IEEE, 2000, pp. 77–83.

[8] V. Almendra and D. Enachescu, "Using self-organizing maps for binary classification with highly imbalanced datasets," *International Journal of Numerical Analysis and Modeling, series b*, vol. 5, no. 3, pp. 238–254, 2014.

[9] A. A. Akinduko and E. M. Mirkes, "Initialization of self-organizing maps: principal components versus random initialization. a case study," *arXiv preprint arXiv:1210.5873*, 2012.

[10] The MathWorks, Inc., "Cluster with self-organizing map neural network," [Accessed: April 15, 2017]. [Online]. Available: https://www.mathworks.com/help/nnet/ug/cluster-with-self-organizing-map-neural-network.html

[11] C. T. U. i. P. Faculty of Electrical Engineering, "newsom - create a self-organising map," [Accessed: April 15, 2017]. [Online]. Available: http://radio.feld.cvut.cz/matlab/toolbox/nnet/newsom.html

[12] S. O'Keefe, "Interpretation of self-organisation," Lecture notes for INCA, Department of Computer Science, University of York, November 2016.