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.

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 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	Yes (1729 Positive + 1729 Negative)	No (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 variance < 0.01, all % variance < 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.

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.

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.

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	HumidityRatio	Numeric	[0.0027, 0.0065]

Figure 6: Input variables to the SOM.

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.