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.

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.

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.