Two Hidden Layers are Usually Better than One

Alan J Thomas[1,*], Miltos Petridis[2], Simon D Walters[1], Saeed Malekshahi Gheytassi[1], and Robert E Morgan[1]

[1]School of Computing Engineering and Mathematics, University of Brighton, United Kingdom
`alan.j.thomas@gmail.com*,{s.d.walters,`
`m.s.malekshahi,r.morgan2}@brighton.ac.uk`
[2]Faculty of Science and Technology, Middlesex University, United Kingdom
`m.petridis@mdx.ac.uk`

Abstract. This study investigates whether feedforward neural networks with two hidden layers generalise better than those with one. In contrast to the existing literature, a method is proposed which allows these networks to be compared empirically on a hidden-node-by-hidden-node basis. This is applied to ten public domain function approximation datasets. Networks with two hidden layers were found to be better generalisers in nine of the ten cases, although the actual degree of improvement is case dependent. The proposed method can be used to rapidly determine whether it is worth considering two hidden layers for a given problem.

Keywords: feedforward neural networks · how many hidden layers · universal function approximation · transformative optimisation · optimal FNN topology · one or two hidden layers

1   Introduction

The most important aspect of the design of a neural network is its structure or topology, since this is crucial to its generalisation capability. In the case of a fully interconnected feedforward neural network (FNN), and given a fixed set of inputs and outputs, the topology is directly determined by the number of hidden nodes and layers. Whilst there is an extraordinary volume of literature on the subject of hidden node selection, there is scarcely any about hidden layer selection. This is almost certainly due in part to proofs that networks with a single hidden layer are sufficient for universal approximation [1–3]. Furthermore, the search space of candidate topologies is linear - so they are easier to find and train. Consequently, there is less interest in neural networks with two or more hidden layers and they are rarely used in practice [4].

However, it has been shown that two-hidden-layer feedforward networks (TLFNs) can outperform single-hidden-layer ones (SLFNs) in some cases. Indeed there is some evidence that certain problems can only be solved with a second hidden layer [5–7]. What is lacking in the literature is any indication about how SLFNs and TLFNs compare in practical situations. To redress this, SLFNs and TLFNs compete head to head on ten public domain datasets. In order to ensure a fair competition, all factors other than the number of hidden layers are kept constant.

In section 2, related work on the subject of the number of hidden layers is discussed. Section 3 describes "transformative optimisation", the approach used in this paper. This is a new name for a technique previously developed by the Authors in [8, 9]. It is ideally suited for the hidden-node-by-hidden-node comparisons used in the experiments. Sections 4 and 5 detail the experimental setup and methods. In section 6, the results are summarised and discussed. It was found that TLFNs showed improvement in generalisation performance over SLFNs in most of the cases.

2 Related Work

The volume of literature comparing SLFNs and TLFNs is very scarce [10]. Funahashi [11] proved that any continuous function can be approximated with a single hidden layer. However it was subsequently shown by Chester that two hidden layers were better when dealing with pinnacle functions [5]. He states that "the problem with a single hidden layer is that the neurons interact with each other globally, making it difficult to improve an approximation at one point without worsening it elsewhere". He goes on to point out that in a TLFN, the first hidden layer can partition the input space into smaller regions, leaving the second hidden layer free to improve the approximation. Sontag [7] approached the question from a different angle. He showed that some nonlinear control systems require two hidden layers to achieve stabilization. Brightwell [6] went on to show that certain classes of problems are not realizable with a single hidden layer: "XOR-situation, XOR-bow-tie, XOR-at-infinity, and critical cycle".

In 2011, Nakama conducted a "fair and systematic" comparison of multiple and single hidden layer networks [10]. These had the same number of inputs, outputs, nodes, and connections, and approximate the same target functions. He shows that the learning rate is more flexible for a single hidden layer (Figure 1(C)), and that it also converges faster than a multiple hidden layer (Figure 1(A)). However, the activation functions used are solely a linear function of the weights a-d. Thus these can be rearranged to reduce both networks to the linear perceptrons shown in Figure 1(B) and 1(D). Therefore the results are possibly subject to misinterpretation, such as justification to exclude multiple hidden layers from investigations [12].
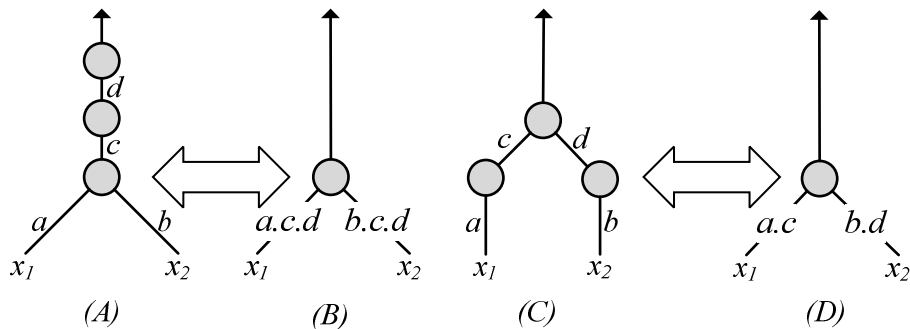

Fig. 1. Nakama Networks – Multiple (A) and single hidden layer (C), with equivalents (B&D)

The current study is empirical in contrast to [5–7]. It is similar to [10] in its aim to compare the networks fairly, although the definition of fairness differs slightly. Here "fairness" is defined as the networks having the same number of hidden nodes, the same activation functions, and trained with the same training algorithm with the same default training parameters. The Levenberg-Marquardt algorithm [13] is used because it is well suited and commonly used for function approximation problems [14]. The current study differs from [10] in several key areas:

- **Network Structure** – Only conventional, fully interconnected feedforward networks are used. These have non-linear activation functions in the hidden nodes, and a linear activation function in the output node. In contrast, the networks in [10] are not fully interconnected and use only linear activation functions. As has been shown, the consequence of this is that the networks can be reduced to linear perceptrons with no hidden layers at all.
- **Method of Comparison** – The generalisation capability of the networks is compared on a hidden-node-by-hidden-node basis over an adequate range of 1–64 nodes. This method of comparison was chosen as it is the main area of concern for designers of neural networks. The node-by-node comparison is facilitated by leveraging transformative optimisation, a technique developed by the Authors further described in section 3. In contrast, [10] concerns itself with the learning rate and convergence properties in two specific cases with a total of 3 nodes.
- **Training Data** – Ten public domain datasets are used. These contain varying amounts of inputs, training samples and noise, as it is important to compare the networks over a wide range of different problems. In contrast [10] considers two simple mathematical functions ($y = x_1 + x_2$ and $y = x_1^2 + x_2^2$).

To the best of the Authors' knowledge, this is the first study of its kind to be undertaken.

3   Transformative Optimisation

Attempting to perform a hidden node for node comparison between SLFNs and TLFNs is a complicated affair for two reasons. Firstly, there are many different ways to allocate a given number of nodes between the first and second hidden layers of a TLFN; and secondly, the number of candidate topologies to be considered for a TLFN is quadratic. Thus it can take a prohibitive amount of time to test all candidates. Transformative optimisation is a new name given to a technique developed by the Authors which can be applied to solve both of these problems. It has been so named because it transforms the space of candidate TLFN topologies from quadratic to linear.

Consider the set of candidate TLFN topologies T, where the number of nodes in each hidden layer varies between 1 and k, i.e.

$$T = \left\{ \tau_{11}, \tau_{21}, \dots, \tau_{ij}, \dots, \tau_{kk} \right\}, \tag{1}$$

where
$$\tau_{ij} = N_0 : i : j : N_3 \tag{2}$$

The constants, N_0 and N_3 represent the number of inputs and outputs respectively for any given domain, and i and j are the number of nodes in the first and second hidden layers respectively. Transformative optimisation works by transforming this set into a different set $T' = \{\tau_4, ..., \tau_{n_h}, ..., \tau_{2k}\}$, where $T' \subset T$. This only has a single degree of freedom, n_h, which represents the total number of hidden nodes. Here

$$\tau_{n_h} = N_0 : [\varrho n_h + c] : [(1 - \varrho)n_h - c] : N_3, \qquad (3)$$

where ϱ is a ratio determining the node allocation between the first and second hidden layers, and c is a constant. It has been shown in the Authors' previous studies [8, 9], that the optimal values are $\varrho = 0.5$ and $c = 1$. These had the highest probability of finding the best generalisers over the same ten datasets used in the current experiments. Substituting these values into (2), we have

$$\tau_{n_h} = N_0 : [0.5n_h + 1] : [0.5n_h - 1] : N_3. \qquad (4)$$

It should be noted that the lowest value of n_h which yields a two hidden layer network is four, organised as three nodes in the first hidden layer and a single node in the second. Additionally, odd values of n_h yield fractional node values. This can be dealt with in two ways. The first rounds the nodes in the first hidden layer down and rounds those in the second up or vice versa. Alternatively, only even values of n_h are considered:

$$\tau_{n_h} = N_0 : [i + 1] : [i - 1] : N_3, \qquad (5)$$

where $n_h = 2i$ and $2 \leq i \leq k$. In this case the set of candidate topologies T' is

$$T' = \{\tau_2, ..., \tau_i, ..., \tau_k\}, n_h = 2i. \qquad (6)$$

Note that from (1), $|T| = k^2$, and from (2), $|T'| = k - 1$. This corresponds to the transformation of the candidate space from quadratic to linear. In this study, the rounding method is used to compare the performance of SLFNs and TLFNs node by node.

4 Experiments

4.1 Data Acquisition

A total of ten datasets were acquired for the experiments. These were selected on the basis of their availability in the public domain, and suitability for function approximation. One consequence of these selection criteria is that the datasets all have a single output. An exception to this was the engine dataset available in Matlab, which has two inputs and two outputs. However, the torque output was reassigned as an input, thus converting it to a 3 input, 1 output dataset. The datasets were sourced from the UCI Machine Learning Repository [15], Bilkent University Function Approximation Repository [16], University of Porto Regression Datasets [17], and Matlab. The datasets used are summarised in Table 1.

Table 1. Dataset Summary

Name	Samples	Inputs	Source
Abalone	4177	8	UCI Machine Learning Repository
Airfoil Self-noise	1503	5	UCI Machine Learning Repository
Chemical	498	8	Matlab chemical_dataset
Concrete	1030	8	UCI Machine Learning Repository
Delta Elevators	9517	6	University of Porto Regression Datasets
Engine	1199	3	Matlab engine_dataset
Kinematics	8292	8	BU Function Approx. Repository
Mortgage	1049	15	BU Function Approx. Repository
Simplefit	94	1	Matlab simplefit_dataset
White Wine	4898	11	UCI Machine Learning Repository

Following the data acquisition phase, Matlab R2014b was used to prepare the data, create and train the neural networks and generate the raw results for analysis.

4.2 Data Preparation

This one-off phase split each dataset into three sub-sets: training, validation and test. The validation set was used to stop the training early when the validation error began to rise, and the test set was used exclusively as an estimate of each network's generalisation error. Eighty percent of the original data was randomly allocated to the training set, and the remaining twenty percent was equally allocated between the validation and test sets. In order to ensure consistency, the same sub-sets were used to create and test all networks within the scope of a given dataset.

4.3 Network Creation, Training and Evaluation

All networks in the experiments were created, trained and evaluated in an identical fashion. They were created using the 'fitnet' function of the Neural Network Toolbox, with the 'mapminmax' function for input and output processing. The activation function of all hidden nodes was 'tansig' and that of the output node was linear. The training function used was Levenberg-Marquardt, 'trainlm', which often yields the best results for function approximation problems. The networks were all trained using Matlab's default training parameters: 1000 epochs, training goal of 0, minimum gradient of 10^{-7}, 6 validation failures, $\mu = 0.001$, $\mu_{dec} = 0.1$, $\mu_{inc} = 10$ and $\mu_{max} = 10^{-10}$.

The error function used during training was the mean squared error function 'mse', however the generalisation error was reported using the normalised root mean squared error (NRMSE). This is a function of the number of samples n, the target output $\hat{y}_i$, and the actual output y_i. It is normalised to the target output swing $\hat{y}_{max} - \hat{y}_{min}$, in order to faciltate comparison between different datasets. The NRMSE ε is given by:

$$\varepsilon = \frac{1}{\hat{y}_{max} - \hat{y}_{min}} \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{7}$$

4.4 Experimental Method

A total of 20 experiments were carried out on each of the datasets. Half of these were using SLFNs, and the other half using TLFNs. Each experiment consisted of varying the number of hidden nodes n_h between 1 and 64. For SLFNs, the number of hidden nodes was simply n_h. For TLFNs, the number of nodes in each of the hidden layers was calculated according to equation (3), using the rounding method described in section 2. Because the random weight initialisation can cause training to getting trapped in local minima [4], each topology is trained 30 times, and the network with the most favourable generalisation error is chosen as the "champion" network for that particular topology. For clarity, the pseudo-code for a single experiment is shown in Figure 2.

```
function e = singleExperiment(type, nh)
  for nh = 1 to 64 do
    if type is 'SLFN'        % calc nodes in each layer
      n1 = nh                % First hidden layer nodes
      n2 = 0                 % SLFN has no hidden layer 2
    elseif type is 'TLFN'
      n1 = int(0.5nh + 1)    % round down for layer 1
      n2 = nh – n1           % Calculate nodes in layer 2
    end if
    for candidate = 1 to 30  do % process 30 networks
      net = createNetwork(n1,n2)
      nrmse[candidate] = trainNetwork(net)
    end do
    e[nh] = min(nrmse) % Calculate winner's error
  end do
  return e
end function
```

Fig. 2. Pseudo-code for a single experiment

5 Results and Discussion

For each experiment, the overall "champion" is the network with the lowest generalisation error. Its total number of hidden nodes n_h, and generalisation error ε are recorded. The results are summarised in Table 2, were $\mu(n_h)$ represents the mean number of hidden nodes and $\mu(\varepsilon)$ the mean generalisation error (as a percentage) over ten experiments. The relative improvement is calculated as $\delta\mu(\varepsilon) = \mu(\varepsilon_{SLFN}) - \mu(\varepsilon_{TLFN})$, and the improvement factor is calculated as $f = \delta\mu(\varepsilon)/\varepsilon_{SLFN}$. The winners are in Table 2 are emboldened.

Table 2. Results Summary

Dataset	$\mu(n_h)$		$\mu(\varepsilon)$ (%)		$\delta\mu(\varepsilon)$	f
	SLFN	**TLFN**	**SLFN**	**TLFN**	*(%)*	*(%)*
Abalone	34.1	24.7	6.5280	**6.4597**	0.0683	1.05
Airfoil	40.9	38.7	3.8244	**2.8222**	1.0022	26.21
Chemical	31.6	36.4	**3.3738**	3.4762	-0.1024	-3.04
Concrete	44.5	41.2	4.0272	**3.6937**	0.3335	8.28
Delta Elevators	10.7	18.6	5.0334	**5.0315**	0.0019	0.04
Engine	56.9	46.9	0.8963	**0.8428**	0.0535	5.97
Kinematics	54.1	35.2	4.5220	**4.3730**	0.1490	3.30
Mortgage	51.7	50.1	0.3497	**0.3424**	0.0073	2.10
Simplefit	10.4	10.8	0.0014	**0.0012**	0.0002	12.63
White Wine	36.3	47.0	11.4743	**11.3983**	0.0761	0.66

As might be expected, the results were varied and dataset dependent. The generalisation error is known to be dependent on the complexity of the function to be approximated as well as the level of noise within it. For example, the Simplefit dataset is a very simple function which has no noise at all. It yielded almost zero generalisation error, and there is not much to be gained by using a TLFN. On the other end of the scale, the Airfoil dataset showed significant gains in generalisation error for a TLFN. Overall, 9 out of 10 datasets showed an improvement of some kind when using two hidden layers. Some showed more significant gains than others. The only exception to this is with the Chemical dataset, where a single hidden layer performed best. However this was only a 0.1% relative improvement $\delta\mu(\varepsilon)$. Since the generalisation error is so low in some cases, it was thought that a fairer method of comparison was using the improvement factor f in the final column of Table 2. The overall average improvement factor was 5.72% with a standard deviation of 8.5%. In over half of the cases, the TLFNs also achieved this improvement with fewer hidden nodes. It should be remembered, however, that node for node TLFNs are more complex as they have more weights. Since it is the weights which learn the problem, TLFNs have a larger storage capacity, which might account for this improvement.

So which is better? The evidence in these experiments point to TLFNs. However, the actual amount of improvement is case dependent. Furthermore, low complexity might be a design consideration or requirement and this might need to be balanced against the potential gains in generalisation error. So perhaps a better question might be: Is it *worth* using a TLFN? Fortunately, through transformative optimisation, this can easily be checked. Although the full 1-64 node scans used in these experiment (Figures 4-6 in Appendix A) each took several hours, this need not be the case. Binary sampling techniques can be used to reduce this process to a matter of minutes, whilst giving a broad idea of the likely gains (if any) and sometimes even an idea of where to look. This is illustrated in Figure 3, which shows the result of binary sampling applied to the Airfoil dataset.
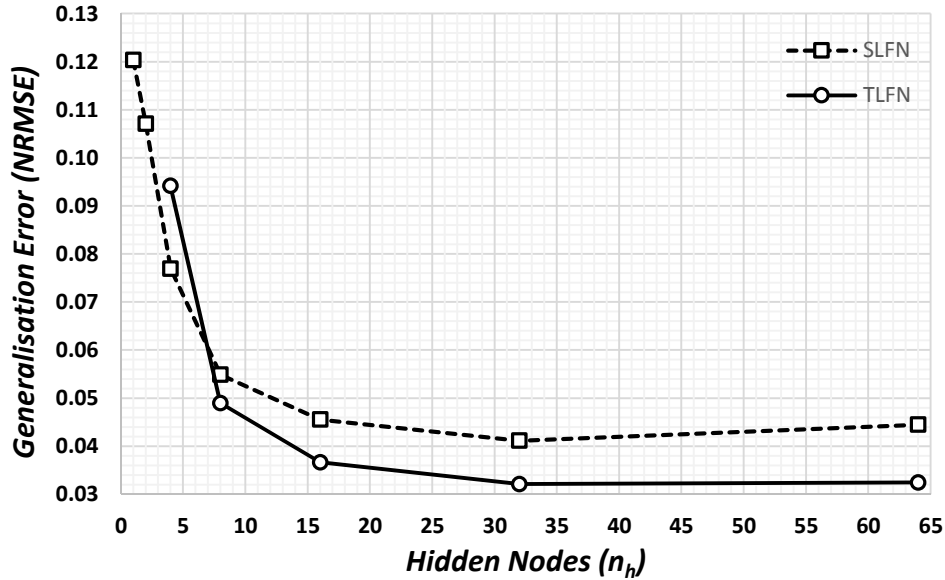
Fig. 3. Airfoil using 12 point binary sampling

6   Conclusion

This study set out to discover whether networks with two hidden layers generalise better than those with a single hidden layer in practical situations. In order to answer this question, a method called "transformative optimisation" was proposed and applied to perform a hidden-node-by-hidden-node comparison of SLFNs and TLFNs across ten separate datasets. It was found that in nine out of ten cases TLFNs outperformed SLFNs, but that the amount of improvement was very case dependent. However, the proposed method can be used in conjunction with binary sampling to rapidly determine whether it is worth using two hidden layers for any given problem. Although the results presented here indicate that TLFNs outperform SLFNs, further investigation with more complicated real-life datasets is necessary.

On a final note, this method could potentially be used for other training algorithms, although it would need to be verified whether the optimal values of $\varrho = 0.5$ and $c = 1$ still hold. Early indications are that this could well be the case for the 'trainscg' training algorithm [8], although more extensive testing is required. This verification, as well as equivalent comparisons of TLFNs and SLFNs for other training algorithms could be the subject of further work.

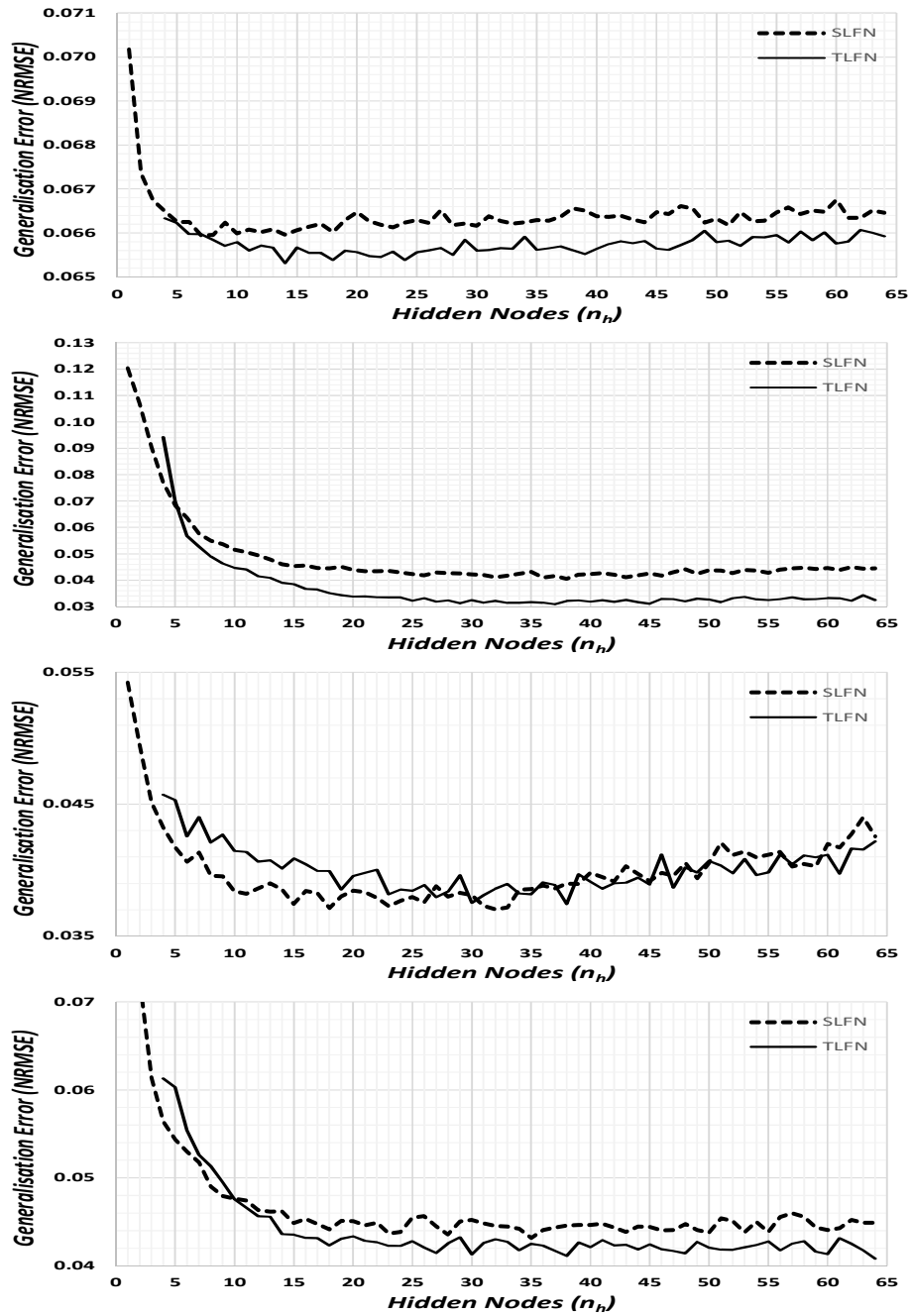# Appendix A – Full Results: Average Node for Node Comparisons

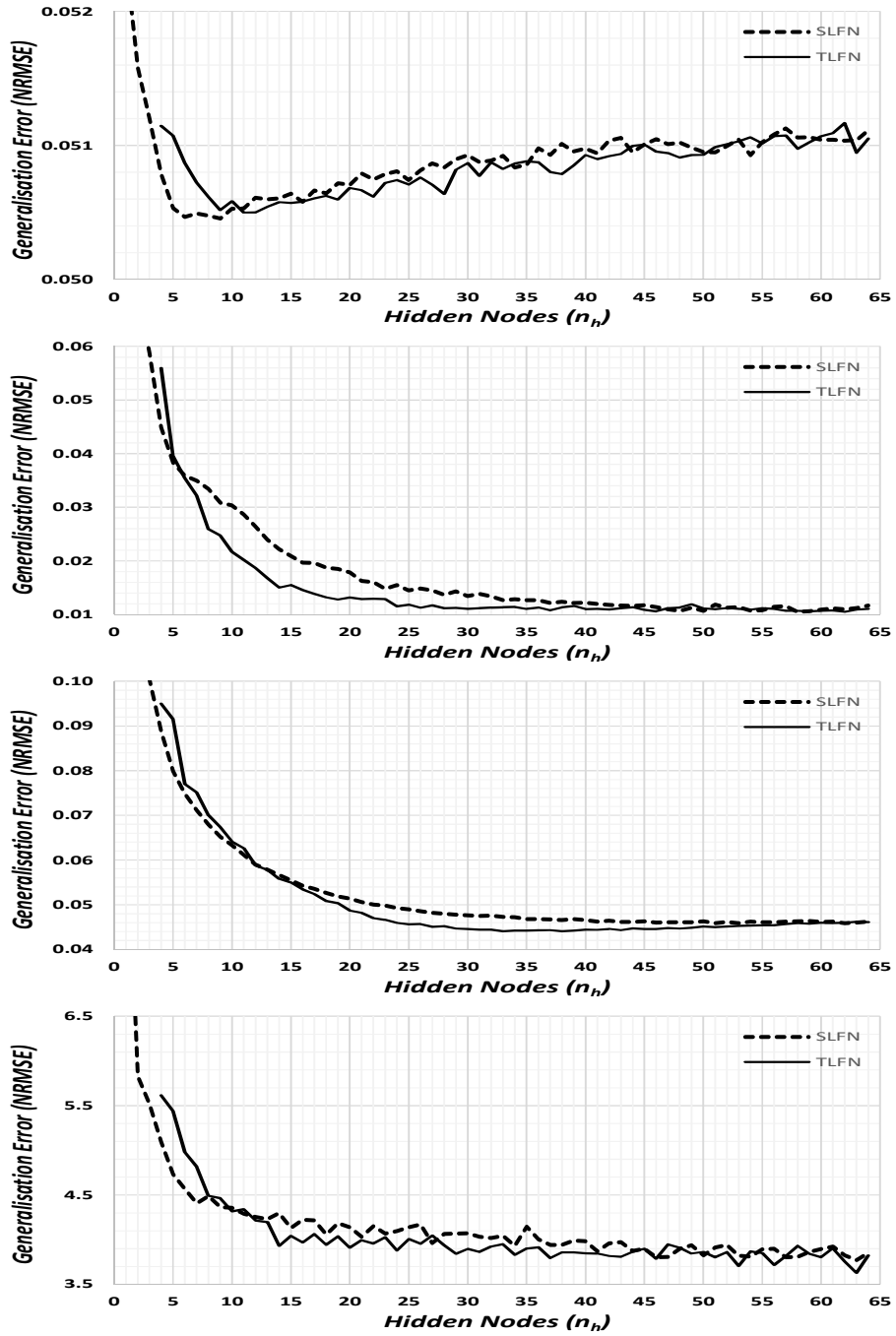**Fig. 4.** Abalone (top), Airfoil, Chemical and Concrete (bottom)

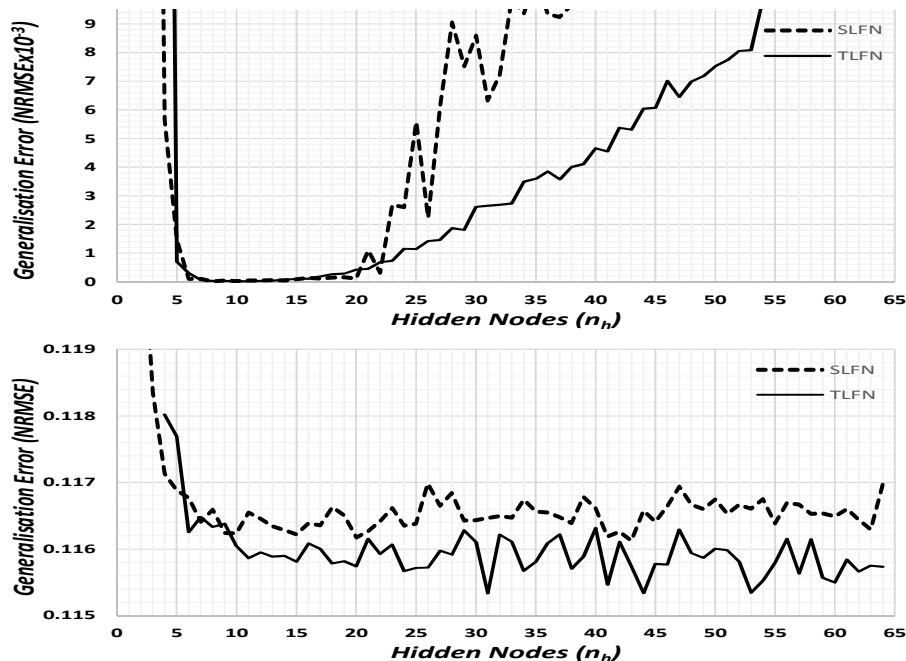**Fig. 5.** Delta Elevators (top), Engine, Kinematics, and Mortgage (bottom)

Fig. 6. Simplefit (top) and White Wine (bottom)

References

1. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**, 359–366 (1989)
2. Hornik, K., Stinchcombe, M., White, H.: Some new results on neural network approximation. Neural Netw. **6**, 1069–1072 (1993)
3. Huang, G.-B., Babri, H.A.: Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. IEEE Trans. Neural Netw. **9**, 224–229 (1998)
4. Zhang, G. P.: Avoiding Pitfalls in Neural Network Research. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **37**, 3–16 (2007)
5. Chester, D. L.: Why two hidden layers are better than one. In: Caudhill, M. (ed.) International Joint Conference on Neural Networks, vol. 1, pp. 265-268. Laurence Erlbaum, New Jersey (1990)
6. Brightwell, G., Kenyon, C., Paugam-Moisy, H.: Multilayer neural networks: one or two hidden layers?. In: Mozer, M. C., Jordan, M. I., Petsche, T. (eds.) Advances in Neural Information Processing Systems. vol. 9, pp. 148–154., MIT Press, Cambridge, MA (1997)
7. Sontag, E., D.: Feedback stabilization using two-hidden-layer nets. IEEE Trans. Neural Netw. **3**, 981–990 (1992)
8. Thomas, A. J., Walters, S. D., Petridis, M., Malekshahi Gheytassi, S., Morgan, R. E.: Accelerated Optimal Topology Search for Two-Hidden-Layer Feedforward Neural Net-works. In: Jayne, C. and Iliadis, L. (eds.) EANN 2016. CCIS, vol. 629, pp. 253–266. Springer, Switzerland (2016). doi: 10.1007/978-3-319-44188-7_19

9. Thomas, A. J., Walters, S. D., Malekshahi Gheytassi, S., Morgan, R. E., Petridis, M.: On the Optimal Node Ratio between Hidden Layers: A Probabilistic Study. Int. J. Mach. Learn. Comput. **6**, 241-247 (2016). doi: 10.18178/ijmlc.2016.6.5.605

10. Nakama, T.: Comparisons of Single- and Multiple-Hidden-Layer Neural Networks. In: Liu, D., Zhang, H., Polycarpou, M., Alippi, C., He, H. (eds.) Advances in Neural Networks – ISNN 2011 Part 1. LNCS, vol. 6675, pp. 270–279. Springer, Heidelberg (2011)

11. Funahashi, K. –I.: On the approximate realization of continuous mappings by neural networks. Neural Netw. **2**, 183–192 (1989)

12. Idler, C.: Pattern Recognition and Machine Learning Techniques for Algorithmic Trading. MA thesis, FernUniversität, Hagen, Germany (2014)

13. Moré, J. J.: The Levenberg-Marquardt algorithm: Implementation and theory. In: Watson G. A. (ed.) Numerical Analysis. LNM, vol. 630, pp. 105–116. Springer, Heidelberg (1978). doi: 10.1007/BFb0067690

14. Beale, M. H., Hagan, M. T., Demuth, H. B.: Neural Network Toolbox User's guide. https://www.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf

15. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/

16. Bilkent University Function Approximation Repository. http://funapp.cs.bilkent.edu.tr/DataSets/

17. Regression Datasets. http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

18. Yeh, I. -C.: Modeling of strength of high performance concrete using artificial neural networks. Cem. Concr. Res., **28**, 1797–1808 (1998)