Accelerated Optimal Topology Search for Two-hidden-layer Feedforward Neural Networks

Alan J Thomas*, Simon D Walters, Miltos Petridis, Saeed Malekshahi Gheytassi, Robert E Morgan

School of Computing Engineering and Mathematics, University of Brighton, United Kingdom.
{a.j.thomas*,s.d.walters,m.petridis,
m.s.malekshahi,r.morgan1}@brighton.ac.uk

Abstract. Two-hidden-layer feedforward neural networks are investigated for the existence of an optimal hidden node ratio. In the experiments, the heuristic $n_1 = int(0.5n_h + 1)$, where n_1 is the number of nodes in the first hidden layer and n_h is the total number of hidden nodes, found networks with generalisation errors, on average, just 0.023%-0.056% greater than those found by exhaustive search. This reduced the complexity of an exhaustive search from quadratic, to linear in n_h, with very little penalty. Further reductions in search complexity to logarithmic could be possible using existing methods developed by the Authors.

Keywords: two-hidden-layer feedforward · ANN · exhaustive search · optimal topology · optimal node ratio · Heurix · universal function approximation

1 Introduction

Function approximators are an important class of artificial neural networks. Since it was shown that multilayer feedforward neural networks with as few as a single hidden layer are universal function approximators [1], they have enjoyed an upsurge of popularity in diverse domains. In the automotive arena, they are used increasingly to predict engine emissions, and typically involve an exhaustive or 'trial and error' search through one or two hidden layers to find the optimal topology - though the former is by far the most common [2, 3, 4, 5]. This could well be because of the prohibitive time required to conduct an exhaustive quadratic search through two hidden layers. This paper addresses the question: 'Does there exist an optimal ratio of nodes between the first and second hidden layers of a two-hidden-layer neural network (TLFN)?' If so, this could be combined with existing network topology optimisation techniques to reduce their complexity. For example, the complexity of an exhaustive search for a TLFN would be reduced from a quadratic search $O(n^2)$ to a linear search $O(n)$, diagonally along the optimal ratio line.

In this paper, a heuristic relationship between the total number of hidden nodes, n_h and the number of nodes in the first hidden layer n_1 is proposed. TLFNs created using this heuristic are compared with the best of those found by searching all possible combinations of nodes in the first and second hidden layers (n_1 and n_2 respectively) such

that $n_h = n_1 + n_2$. Although this heuristic is not guaranteed to produce the best node ratio, in our experiments the generalisation error is only $0.023\% - 0.056\%$ greater than the best of any other node combination.

2 Problem Description

When designing a feedforward neural network, the number of inputs and outputs are easily selected as these are determined by the application. The number of hidden layers required depends on the complexity of the function. For functions which are linearly separable, no hidden layers are required at all. Given a sufficiently large number of hidden units, a single layer will suffice [1], however two hidden layers can often achieve better result than a single layer [6]. In the Authors' own experience, node for node, a TLFN will give a better generalisation capability than an single-hidden-layer feedforward neural network (SLFN) in many cases.

The most challenging and time consuming aspect of the design is choosing the optimal number of hidden nodes. It is assumed here that 'optimal' means 'yielding the best generalisation capability'. Too few hidden nodes, and the network simply will not have the capacity to solve the problem. Conversely, too many, and the network will memorise noise within the training data, leading to poor generalisation capability. Thus the challenge is finding a network which achieves the best balance, ideally in a reasonable time.

3 Related Work

Many optimisation techniques for feedforward neural networks have been proposed in the literature. These can be broadly summarised as:

3.1 Rules of Thumb

These are generally associated with guessing the best number of hidden nodes for single-hidden-layer feedforward networks (SLFNs). There do not appear to be any that pertain to TLFNs.

3.2 Trial and Error

This is a very primitive approach likely to yield extremely sub-optimal results. However, this term is occasionally applied to an exhaustive search between certain bounds. In [5], for example, the term is used to describe the search for a TLFN which varies the number of nodes in each hidden layer between 1 and 20, with a resulting search space of 400 different topologies.

3.3 Exhaustive Search

This involves training networks with every possible combination of hidden nodes between 1 and some upper bound, N_h, and choosing the network with the best generalisation performance. Huang and Babri rigorously proved that an SLFN with at most N_h hidden neurons can learn N_s distinct samples with zero error [7]. Though this gives us an upper bound on the number of hidden neurons, it also means that at this bound the network will also overfit by exactly learning the noise within the training set. Thus an exhaustive search for an SLFN should vary the number of hidden nodes from 1 up to an absolute maximum of $N_h = N_s$.

Huang later proved that the upper bound on the number of hidden nodes N_h for TLFNs with sigmoid activation function is given by $N_h = 2\sqrt{(n_3 + 2)N_s}$, where n_3 is the number of outputs. These can learn at least N_s distinct samples with any degree of precision [8]. Interestingly, Huang also demonstrated that the storage capacity can be increased by reducing the number of outputs, which is probably the best argument for limiting the number of outputs for function approximation to a single output. With that in mind, substituting $n_3 = 1$ we have $N_h = \sqrt{12N_s}$. This is $\sqrt{N_s/12}$ times lower than that of an SLFN. This means that an exhaustive search for a TLFN with for example 10,000 training samples, would have an upper bound which is 29 times lower than for an SLFN.

With exhaustive searches, several networks of each topology need to be trained to filter out networks where the initial random weight allocation might cause the training to get trapped in local minima. Because of this effect, it is unlikely that the actual global optimum will be found. Since this depends on all the weights being exactly correct, the probability of finding the global optimum will increase with the number of weights. The result is that the 'optimal' topology returned by an exhaustive search will be different on successive searches. For SLFNs, the complexity of an exhaustive search is linear $O(n)$, whereas for TLFNs, it is quadratic $O(n^2)$.

3.4 Growing Algorithms

At their simplest, these are similar to exhaustive searches. They generally start with a single hidden node, and increase the number of hidden nodes one by one until the improvement in generalisation error is negligible. Using this approach with TLFNs is problematic because the sudden variation of node ratio on new rows will result in spikes on the generalisation landscape resulting in premature termination. Other types of growing algorithms combine simultaneous growing and training. These can be classified as non-evolutionary [9], and evolutionary [10]. The latter are notoriously time consuming.

3.5 Pruning Algorithms

With this approach, an oversized network is trained and the relative importance of the weights subsequently analysed. The least important weights are removed and the network retrained. The problem with these is determining what constitutes an oversized network in the first instance, and their time complexity. Brute force approaches which

set each weight in turn to zero and eliminates it if it has a negligible effect on the generalisation error. These have a complexity of $O(N_s w^3)$, where N_s is the number of samples in the training set, and w is the number of weights in the original oversized network [11].

3.6 Heuristic Algorithms

These estimate the optimal number of hidden nodes by sampling a sub-set of topologies and using curve fitting techniques to predict the optimum topology. A system previously developed by the Authors [12] can create SLFNs with a generalisation error of as little as 0.4% greater than those found by exhaustive search with a complexity of $O(log_2(n))$.

3.7 Proposed Method

This is not a separate method *per se*, but rather a heuristic to be used in conjunction with another optimisation method. If there exists an 'optimal' node ratio for a TLFN, then it effectively reduces its complexity to that of an SLFN.

4 Experiments

All experiments were carried out using the Matlab R2014b environment. The networks were created using the Neural Network Toolbox 'fitnet' function to generate the SLFNs and TLFNs where appropriate. Two separate datasets were used, with different numbers of inputs. These were trained the Levenberg-Marquardt training function, 'trainlm' which is commonly used for function approximation as it has often been found to yield the best results [2, 3, 4, 5]. For comparison, the second dataset was also trained with the Scaled Conjugate Gradient training function 'trainscg'.

4.1 Data Preparation

The datasets were chosen because of their availability in the public domain, allowing the findings to be independently verified. In all cases, the data is split into three subsets: Training (80%), Validation (10%) and Test (10%). The Validation set is used to stop the training process when the validation error starts to rise, and the Test set is used exclusively as an estimate of the generalisation error.

For any given dataset, exactly the same subsets were used for every single network created in the experiment. By eliminating any bias in the error surface that may have resulted from a different random split for each network, it was ensured that they were all competing on the same playing field. The only random element at play was thus the initial randomisation of the weights. This initial starting point determines which local minimum in the error surface the training might get stuck in and thus has a direct impact on the generalisation error. For complex error surfaces, it is extremely unlikely that the global minimum will be found.

Dataset 1. The 'engine_data' (available in Matlab), consists of 1199 samples organised as two inputs (fuel and speed) and two targets (torque and NOx). These were reorganised to use torque as a third input, with a single output, NOx. They were subsequently split into Training, Validation and Test subsets (959, 120, and 120 samples, respectively).

Dataset 2. The NASA Airfoil Self-Noise dataset, available from the UCI Machine Learning Repository [13]. This consists of 1503 samples with five inputs: Frequency (Hz), Angle of attack (°), Chord length (m), Free-stream velocity (m/s), and Suction side displacement thickness (m). It has a single output, scaled sound pressure level (dB). These were split into Training, Validation and Test subsets (1201, 151 and 151 samples, respectively).

4.2 Training Algorithms

In all cases, data preprocessing was 'mapminmax' for both inputs and outputs, the transfer function was 'tansig' and the error function for training was 'mse'. However, the generalisation error in the experiments was reported as the normalised root mean squared error (NRMSE), which is given by:

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

where N_s represents the number of samples, $\hat{y}_i$ is the target value, and y_i is the actual value.

Training Algorithm 1. Levenberg-Marquardt training algorithm 'trainlm' using 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 $\mu_{max} = 10^{-10}$.

Training Algorithm 2. Scaled Conjugate Gradient training algorithm 'trainscg' using default parameters: 1000 epochs, training goal of 0, minimum gradient of 10^{-6}, 6 validation failures, $\sigma = 5 \times 10^{-5}$, and $\lambda = 5 \times 10^{-7}$.

5 Experimental Method

Three separate domains were tested:

- **Domain 1** - Dataset 1 using Training Algorithm 1, with the generalisation error averaged over 100 rounds of Fig. 1.

- **Domain 2** – Dataset 2 using Training Algorithm 1, with the generalisation error averaged over 100 rounds of Fig. 1.
- **Domain 3** – Dataset 2 using Training Algorithm 2, with the generalisation error averaged over 300 rounds of Fig. 1. The number of rounds were increased here because of the higher variance in generalisation error when using Algorithm 2.

Within these domains, a number of experiments were carried out each with a constant total number of hidden nodes:

$$n_h = n_1 + n_2 \tag{2}$$

where n_1 and n_2 are the number of nodes in hidden layers 1 and 2 respectively. The values of n_h chosen for these experiments were given by the set:

$$n_h = \{34, 20, 16, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3\}\,. \tag{3}$$

For each value of n_h, TLFNs were created using all possible combinations of n_1 and n_2 satisfying (2). For example if $n_h = 4$, $n_1 = \{1,2,3\}$ and $n_2 = \{3,2,1\}$. This yielded 3 possible networks with topologies $n_0{:}\,1{:}\,3{:}\,n_3$, $n_0{:}\,2{:}\,2{:}\,n_3$ and $n_0{:}\,3{:}\,1{:}\,n_3$. To reduce the effect that the random initial weights have on the generalisation error, 100 networks of each topology were created, and the NRMSE of each calculated from (1). The generalisation errors of the best generalisers (those with the minimum NRMSE on the test set) formed the results of a single round. This is shown in the pseudo-code in Fig. 1, which from any given n_h, returns an array, e, length $n_h - 1$, indexed by the number of nodes in the first hidden layer.

```
function e = singleRound(nh)
  for n1 = 1 to nh-1 do
    n2 = nh - n1           % Calculate n2

    % create and train 100 networks recording NRMSE
    for run = 1 to 100 do
      net = createNetwork(n1,n2)
      nrmse[run] = trainNetwork(net)
    end do

    e[n1] = min(nrmse)     % Calculate winner's error
  end do
  return e
end function
```

Fig. 1. Pseudo-code for a single round

The array, e, was then averaged over 100, 100, and 300 of these rounds for domains 1, 2 and 3, respectively. This was repeated for every value of n_h within the set defined by (3).

6 Results and Discussion

6.1 Optimal Node Ratio Investigation

The results were not at all as expected. In the preliminary investigative experiments with Domain 1, the median NRMSE was used instead of the minimum to determine the winner, and 200 rounds were used. The contents of the arrays, e, were displayed along the y-axis, and their indices (representing the values of n_1) were displayed as an offset from $0.5n_h$ along the x-axis. In other words, the x-axis = $n_1 - 0.5n_h$ as this was where the optimum, if it existed, was expected to lie. However, there seemed a marked symmetry in the region bounded by $n_1 = 4$ and $n_2 = 2$ (narrow dash) and centred on $0.5n_h + 1$ (wide dash) as shown in Fig. 2. From left to centre, this shows a series of contour lines of decreasing constant values of n_h represented by the set in (3). The 'sweet spot' seemed to imply that the number of nodes in the first hidden layer should be greater than 3 and those in the second layer should be greater than 1. Since the dataset had 3 inputs and 1 output, it was suspected at this stage that the sweet spot might be governed by the number of inputs and outputs.

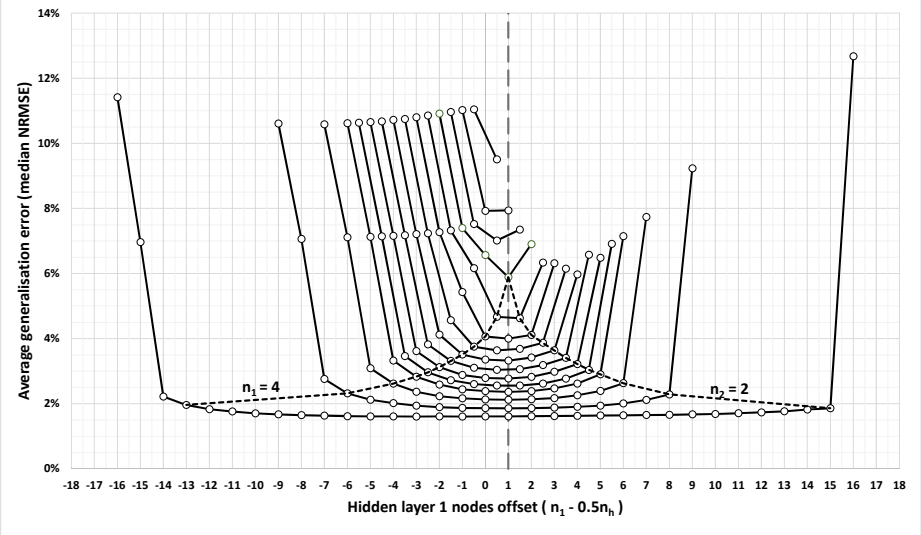

Fig. 2. Initial Domain 1 experiments using average median NRMSE

The main investigation tested whether $n_1 = 0.5n_h + 1$ could also be used to describe the optimum for other datasets and training algorithms. Since n_h can be either even or odd, rounding down was used as a heuristic for the optimal value of n_1, i.e:

$$n_{1(\text{opt})} = int(0.5n_h + 1) \tag{4}$$

As a measure of the accuracy of this prediction, the root mean square difference (rmsd) between the observed minimum generalisation errors, and those obtained using node ratio (4) were calculated. In the preliminary case above, this is less than 0.011%.

In the main body of experiments, each round searched for the networks with the minimum NRMSE (as described in Fig. 1). In this respect, a single round was more representative of an actual exhaustive search for the best generaliser, and multiple rounds could be considered as multiple exhaustive searches. The results, which are shown graphically in Figs. 3-5, show the averages over multiple exhaustive searches (100 for Domains 1-2, and 300 for Domain 3).

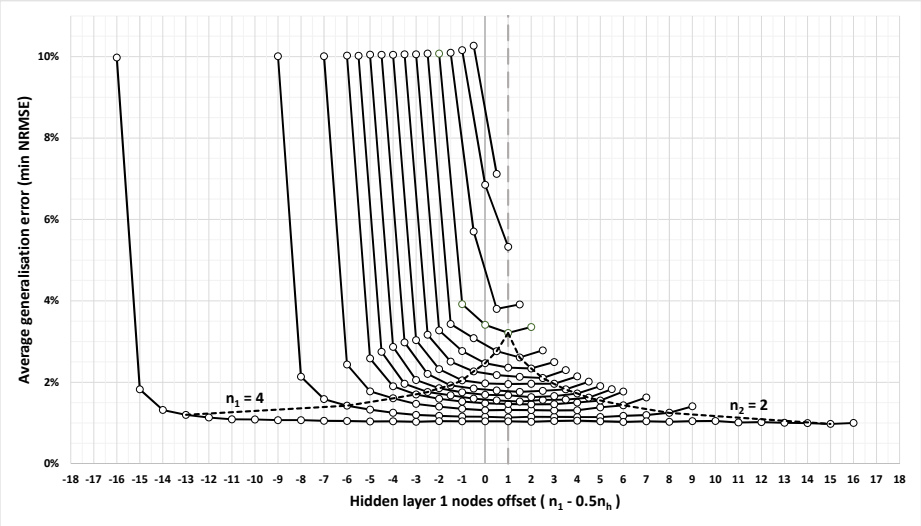

Fig. 3. Domain 1 - Engine Data with Trainlm

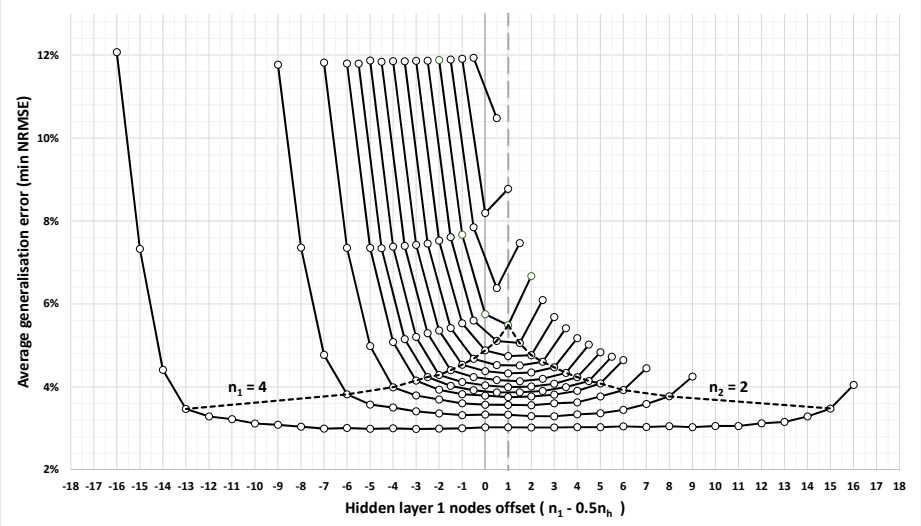

Fig. 4. Domain 2 - Airfoil Self-Noise with Trainlm

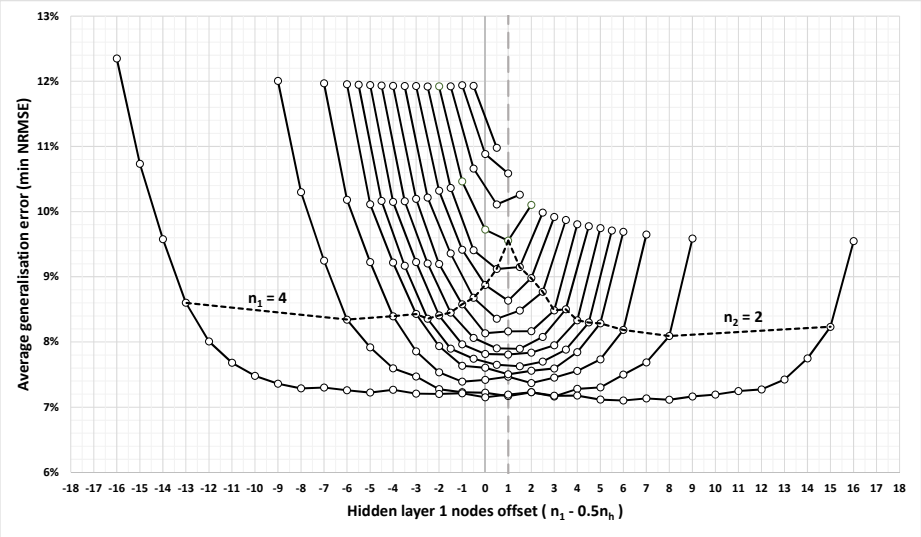

Fig. 5. Domain 3 – Airfoil Self-Noise with Trainscg

The sweet spot is still there in all three cases, although for Domain 1, it not as clearly defined as in the initial experiments, and it seems to lean slightly to the right. Since the number of inputs, the data, and the training algorithm have varied across the three domains, it seems independent of all three within the scope of this investigation.

Table 1. Comparison of Predicted and Observed Best Generalisers.

n_h	Domain 1 NRMSE (%)			Domain 2 NRMSE (%)			Domain 3 NRMSE (%)		
	e_{opt}	e_{min}	δ_e	e_{opt}	e_{min}	δ_e	e_{opt}	e_{min}	δ_e
3	7.12	7.12	0	10.48	10.48	0	10.98	10.98	0
4	5.33	5.33	0	8.77	8.19	0.584	10.59	10.59	0
5	3.80	3.80	0	6.38	6.38	0	10.11	10.11	0
6	3.21	3.21	0	5.49	5.49	0	9.55	9.55	0
7	2.76	2.61	0.155	5.11	5.06	0.049	9.12	9.12	0
8	2.36	2.34	0.022	4.74	4.74	0	8.63	8.63	0
9	2.18	2.10	0.070	4.53	4.52	0.008	8.35	8.35	0
10	1.95	1.95	0	4.33	4.33	0	8.16	8.13	0.0270
11	1.80	1.77	0.032	4.16	4.14	0.026	7.90	7.90	0.0074
12	1.68	1.64	0.042	4.00	4.00	0	7.81	7.81	0
13	1.55	1.53	0.027	3.87	3.87	0	7.65	7.63	0.0224
14	1.47	1.46	0.012	3.75	3.75	0	7.50	7.50	0
16	1.32	1.31	0.014	3.57	3.56	0.008	7.47	7.37	0.0993
20	1.13	1.13	0	3.33	3.29	0.035	7.17	7.16	0.0049
34	1.04	0.98	0.067	3.03	2.98	0.042	7.19	7.10	0.0894

In Table 1, for each value of n_h the average minimum generalisation errors are listed as a percentage for each of the three domains. In this table e_{opt} represents the error at the optimum number of nodes calculated from (4), e_{min} is the observed minimum generalisation error obtained. The error difference is also shown, where $\delta e = e_{opt} - e_{min}$. The root mean square difference (rmsd) between these are shown in Table 2. Since there is an outlier in Domain 2 for $n_h = 4$, which is outside the sweet spot, the rmsd solely within the sweet spot is also included.

Table 2. rmsd between Predicted and Observed Best Generalisers

Domain	**Domain 1**	**Domain 2**	**Domain 3**
rmsd (all)	0.050%	0.152%	0.036%
rmsd (sweet spot)	0.056%	0.023%	0.040%

These results show that although a linear search along (4) is not guaranteed to find the best generalisers (but then neither is a an exhaustive quadratic search), it will find networks within 0.023% - 0.056% of these on average.

6.2 Comparison with SLFNs

In this section, the performance of the TLFNs using the optimal node ratio described by (4) were compared with SLFNs with the same number of hidden nodes for each of the three Domains. The results are shown in Fig. 6.

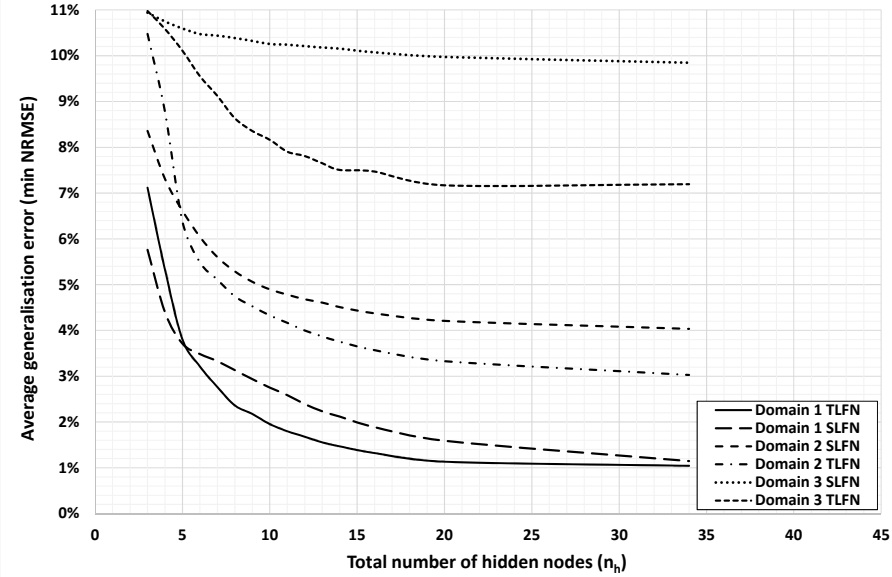

Fig. 6. Node for Node Comparison of TLFNs and SLFNs.

In all three domains, there are advantages to using an optimal TLFN over an SLFN with the same number of hidden nodes. However, for Domain 1, it appears that the generalisation errors are about to converge for $n_h > 34$ nodes. In Domains 2 and 3, there is no sign of any convergence within the scope of the experiments. The greatest gain in generalisation error was for Domain 3, which uses the Scaled Conjugate Gradient algorithm. This is a popular training algorithm for larger numbers of hidden nodes as it is much faster than the Levenberg-Marquardt algorithm, since the latter's training time scales exponentially with the number of hidden nodes. An interesting feature is that for Domains 1 and 2, which both use the Levenberg-Marquardt algorithm, the generalisation errors cross over at $n_h = 5$, above which TLFNs outperform SLFNs. Coincidentally, $n_h = 6$ is the apex of the perceived sweet spot in these experiments. It is unclear at this stage whether the two are related.

7 Conclusions and Further Work

This paper set out to answer the question 'Does there exist an optimal ratio of nodes between the first and second hidden layers of a two-hidden-layer neural network (TLFN)?' Based on the domains tested in the investigation, for $n_h > 5$, this can be described by the relationship $n_1 = 0.5n_h + 1$, or alternatively $n_1 = n_2 + 2$. However, a broader investigation of this hypothesis with different domains is recommended as the subject of future work. In the course of this investigation, a linear search through n_h using the heuristic: $n_1 = int(0.5n_h + 1), n_2 = n_h - n_1$, found networks with a generalisation error of, on average, as little as 0.023% to 0.056% greater than that of the best generalisers. Although this heuristic did not guarantee that the absolute best generaliser was found, neither would a quadratic search through two hidden layers. If this is backed up by further investigation, the implication is that a quadratic search $O(n^2)$ through n_1 and n_2 can be reduced to a linear search $O(n)$ through n_h. This is a very attractive proposition for several reasons:

1. First and foremost the search time would be dramatically reduced and would perhaps encourage engineers to use TLFNs more often.
2. TLFNs can often outperform SLFNs, as was proved in [6], and demonstrated in these experiments.
3. The upper bound on the number of hidden nodes for a TLFN can be much lower than that for an SLFN, as proved in [8]. In fact, it is $\sqrt{N_s/12}$ times lower. This is quite significant, especially for large N_s. For example, this represents a factor of 29 for $N_s = 10{,}000$, meaning 29 times fewer candidates need to be tested.

Given the existence of an optimal ratio, could the search complexity be reduced still further? In a previous paper [12], the Authors have shown that for SLFNs, it is possible to reduce a linear search $O(n)$ to a logarithmic search $O(log_2(n))$. This is achieved by sampling the generalisation error at node values $n_k = 2^k, 0 \leq k \leq N_s$, fitting an error curve of the form $e(n_h) = a{n_h}^{-b} + c$ to these samples, and calculating the optimal number of hidden nodes from its gradient. The choice of gradient determines whether the network is optimised for speed, accuracy or both.

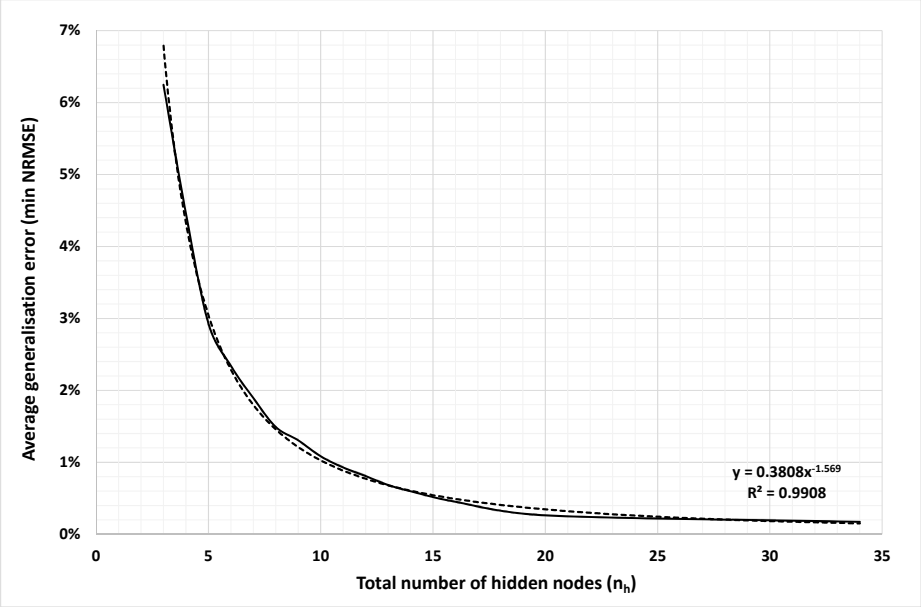

Fig. 7. Domain 1 TLFN with -0.87% offset

In order for this method to be suitable for TLFNs, their generalisation error must also follow similar power law curves. The easiest way to check this is to subtract an offset from the generalisation error and use the trend line feature of the spreadsheet to fit a power law curve. The offset is adjusted to achieve the best value of R^2. Fig. 7 shows the end of this process for Domain 1. This shows that the generalisation error can be described by $e_{opt} = 0.3808{n_h}^{-1.569} + 0.0087$, with a Pearson's correlation coefficient of $R^2 = 0.9908$ or $R = 0.9954$. A similar process was carried out on Domains 2 and 3. The results are summarised in Table 3.

Table 3. Curve fitting variables for the three domains.

Domain	a	b	c	R^2	R
1	0.3808	-1.569	0.0087	0.9908	0.9954
2	0.3854	-1.424	0.0280	0.9936	0.9968
3	0.2407	-1.329	0.0690	0.9420	0.9706

The results are excellent for Domains 1 and 2, which use the Levenberg-Marquardt training algorithm. This is good news, since this training algorithm yields the best generalisation error. Based on the experiments carried out in this paper, the Authors are quite confident that the Heurix system they previously developed [12] will also be suitable for TLFNs. Subject to further work, this method could also be used to find near-optimal TLFNs automatically, with a search complexity of as little as $O(log_2(n))$.

8 Addendum

Since the initial submission of the paper for review, three further domains have been tested using n_h= {1 to14, 16, 20 and 34}, for 100 rounds each. The results are summarised in Table 4.

Table 4. Summary of Further Experiments

Dataset Name	Available	Inputs	Outputs	Samples	Training	rmsd%
chemical_dataset	Matlab	8	1	498	trainlm	0.17
					trainscg	0.06
delta.elevators	Github[1]	6	1	9,517	trainscg	0.09

These datasets are quite interesting. With the former, over the range tested, SLFNs outperform TLFNs with respect to their genaralisation capability. In the case of the latter, there is little or no advantage over a network with no hidden nodes at all. Whilst the heuristic does still yield reasonable results, this does tend to suggest that cases like these ought to be tested for in order to obtain efficient network response times from stimulus to output. This will be the subject of further work.

Acknowledgements. We thank Prof. Martin T. Hagan of Oklahoma State University for kindly donating the Engine Data Set used in this paper to Matlab. We would also like to thank Dr. Roberto Lopez of Intelnics (robertolopez@intelnics.com) for donating the Airfoil Self-Noise dataset; also the dataset's creators: Thomas F. Brooks, D. Stuart Pope and Michael A. Marcolini of NASA.

References

1. Hornik, K., Stinchcombe, M., White, H.: Multilayer Feedforward Networks are Universal Approximators. Neural Netw. 2, 359–366 (1989)
2. Mocanu, F.: On-Board Fuel Identification using Artificial Neural Networks. Int. J. Engines. 7, 937–946 (2014)
3. Yap, W.K., Ho, T., Karri, V.: Exhaust Emissions Control and Engine Parameters optimization using Artificial Neural Network Virtual Sensors for a Hydrogen-powered Vehicle. Int. J. of Hydrogen Energy. 37, 8704–8715 (2012)
4. Roy, S., Banerjee, R., Das, A.K., Bose, P.K.: Development of an ANN based system identification tool to estimate the performance-emission characteristics of a CRDI assisted CNG dual fuel diesel engine. J. Nat. Gas Sci. Eng. 21, 147-158 (2014).
5. Taghavifar, H., Taghavifar, H., Mardani, A., Mohebbi, A.: Modeling the impact of in-cylinder combustion parameters of DI engines on soot and NOx emissions at rated EGR levels using ANN approach. Energy Convers Manag. 87, 1–9 (2014)
6. Chester, D.L.: Why Two Hidden Layers are Better than One. In: International Joint Conference on Neural Networks, vol. 1, pp. 265–268. Laurence Erblaum, New Jersey, (1990)

[1] https://github.com/renatopp/arff-datasets/blob/master/regression/delta.elevators.arff

7. 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. On Neural Netw. IEEE Trans On 9, 224–229 (1989)
8. Huang, G-B.: Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks. IEEE Trans. On Neural Netw. 14, 274–281 (2003)
9. Kwok, T-Y., Yeung, D-Y.: Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. IEEE Trans. On Neural Netw. 8, 630–645 (1997)
10. Azzini, A., Tettamanzi, A.G.B.: Evolutionary ANNs: A state of the art survey. Intell. Artif. 5, 19–35 (2011)
11. Reed, R.: Pruning Algorithms – A Survey. IEEE Trans. On Neural Netw. 4, 740–747 (1993)
12. Thomas, A.J., Petridis, M., Walters, S.D., Malekshahi Gheytassi, S., Morgan, R.E.: On Predicting the Optimal Number of Hidden Nodes. In: 2015 International Conference on Computational Science and Computational Intelligence, pp. 565–570. IEEE CPS (2015)
13. UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml`