On Predicting the Optimal Number of Hidden Nodes

Alan J Thomas, Miltos Petridis, Simon D Walters, Saeed Malekshahi Gheytassi, Robert E Morgan
School of Computing, Engineering, and Mathematics
University of Brighton
United Kingdom
A.J.Thomas@brighton.ac.uk

***Abstract*— Determining the optimal number of hidden nodes is the most challenging aspect of Artificial Neural Network (ANN) design. To date, there are still no reliable methods of determining this a priori, as it depends on so many domain-specific factors. Current methods which take these into account, such as exhaustive search, growing and pruning and evolutionary algorithms are not only inexact, but also extremely time consuming – in some cases prohibitively so. A novel approach embodied in a system called Heurix is introduced. This rapidly predicts the optimal number of hidden nodes from a small number of sample topologies. It can be configured to favour speed (low complexity), accuracy, or a balance between the two. Single hidden layer feedforward networks (SLFNs) can be built twenty times faster, and with a generalisation error of as little as 0.4% greater than those found by exhaustive search.**

Keywords- Feedforward Artificial Neural Network; architecture selection; number of hidden nodes; universal function approximation; Heurix

I. INTRODUCTION

The most challenging aspect of any ANN design is choosing (or fixing) the optimal number of hidden nodes. Though the upper bounds for these have been mathematically proved for function approximation [1], [2]; at these bounds the training set will be exactly memorised including any noise within it. This is detrimental to the ability of the network to generalise the function for unseen input data. Reducing the number of hidden nodes improves generalisation, but going too far will result in a network without the capacity to solve the problem. In practice therefore, optimal means the 'correct' balance between number of hidden nodes and generalisation error.

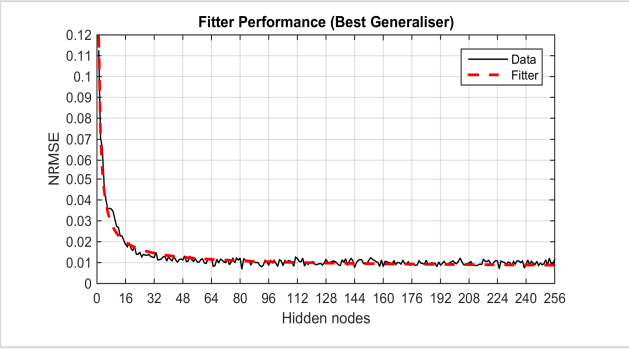

Figure 1. Fitted Error Curve

Regularisation schemes which limit the size of the weights, such as early stopping or weight decay allow oversized networks to be trained without compromising generalisation [3]. If perfect regularisation is assumed, as the number of hidden nodes n is increased, the error $\varepsilon(n)$ will eventually hit a floor level ε_{min}. This will be a function of the training data, the amount of noise within it, the training algorithm, initial random weights etc. Figure 1 shows this effect for a SLFN trained with the Levenberg-Marquardt training algorithm using early stopping. The error function $\varepsilon(n)$ follows a classic power law curve $\varepsilon(n) = cn^b + a$, where n is the number of hidden nodes; $c,\ b < 0,$ and $a = \varepsilon_{min}$ are constants.

This paper explores the theoretical bounds of function approximators and current practical methods to build them in sections II and III respectively. In section IV a novel system called Heurix is introduced which predicts the optimal number of hidden nodes from this error function $\varepsilon(n)$. Its accuracy and performance are compared to exhaustive search in section V, and the paper closes with concluding remarks in section VI.

II. THEORECTICAL BOUNDS OF FUNCTION APPROXIMATION

In 1957, Kolmogorov proved a general representation of Hilbert's 13th problem [4]. Kolmogorov's Superposition Theorem, as it is now known, states that any real-valued, continuous function f defined on an n-dimensional unit cube can be represented as the sum of continuous functions of a single variable. Following refinements by Lorentz [5] and Sprecher [6], the general form is:

$$f(x_1, \dots x_n) = \sum_{q=0}^{2n} \chi\left(q + \sum_{p=1}^{n} \lambda^p \psi(x_p + \epsilon q)\right),$$

where $n \geq 2, \chi$ and ψ are functions of a single variable and λ is a constant. Note that only χ depends on the function, ψ *and* λ depend only on the number of dimensions. Noticing that this essentially describes a 3 layer neural network, Hecht-Nielsen [7] rephrased it to state that any continuous function f defined on an n-dimensional unit cube can be implemented exactly by a 3 layer network with $2n + 1$ hidden nodes. This has been grossly misinterpreted and misapplied by some authors – for example *"you will never require more than twice the number of hidden units as you have inputs"* [8, p. 53]. The fact is that the functions χ and ψ

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.33

are highly non-smooth unlike the more commonly used sigmoidal types. Thus although his theorem demonstrated the theoretical power of neural networks, Hecht-Nielsen summarised it as follows: *"The proof of the theorem is not constructive, so it does not tell us how to determine these quantities. It is strictly an existence theorem. It tells us that such a three-layer mapping network must exist, but it doesn't tell us how to find it. Unfortunately, there does not appear too much hope that a method for finding the Kolmogorov network will be developed soon."* [20, p. 123] .

Although they do not address the bounds on the number of hidden nodes required, a significant result was the proof by Hornik et al. [10] that standard multilayer feedforward networks are universal approximators. Given sufficient hidden nodes with arbitrary squashing functions, these are capable of representing any function to an arbitrary degree of accuracy.

Addressing the issue of the non-smoothness of χ and ψ, in Kolmogorov networks, Kůrková [11] showed that it is possible to approximate these arbitrarily well using any sigmoidal type. In addition, by loosening the requirement from exact function representation to function approximation with error ε, by a two hidden layer feedforward network (TLFN) with $nm(m+1)$ neurons in the first hidden layer, and $m^2(m+1)^n$ neurons in the second hidden layer, where n is the number of inputs and m depends in a complicated fashion on the error ε and function f. This proves that TLFNs can be used to approximate Kolmogorov networks, although *"upper estimates of number of hidden units needed for good approximations of general continuous functions are very large"* [11, p. 503].

More recently, Huang and Babri lowered these bounds, proving that an SLFN with at most N_h hidden neurons can learn N_s distinct samples with zero error. This is true for any bounded, non-linear activation function which has a limit at one infinity [1]. Thus the upper bound for SLFNs is

$$N_h \leq N_s. \tag{1}$$

Huang later extended the work of Tamura and Tateshi [12] to rigorously prove that the upper bound on the number of hidden nodes N_h for TLFNs with sigmoid activation function is given by equation (2), where N_o is the number of outputs. These can learn at least N_s distinct samples with any degree of precision [2].

$$N_h \leq 2\sqrt{(N_o+2)N_s} \tag{2}$$

It is well known that functions which are linearly separable require no hidden nodes at all. For the rest, (1) and (2) prove that within the constraints of their respective theorems, a function of *any* complexity can be reproduced with any degree of precision within the specified bounds. However, if the training set contains noise, as is the case in most practical situations, exactly reproducing the training set will guarantee overfitting. This is extremely undesirable as it is detrimental to the network's generalisation performance. In practice therefore, the number of hidden nodes must necessarily be lower than these bounds and will depend on a number of domain specific factors: including the function complexity, number of inputs and outputs and most crucially, the number of training samples and level of noise within them. The generalisation performance, the measure by which the network will ultimately be judged, is thus domain specific and can only be done *in-situ*, and (1) and (2) suggest that all candidates within these bounds should be considered. Since exhaustive searches can in some cases be extremely time consuming (if not infeasible), a number of alternative approaches, described in section III, have been taken.

III. Building Practical Function Approximators

Current methods for fixing the optimal number of nodes can be broadly classified in the following approaches:

A. Rules of Thumb

Over the years, many *blind* rules of thumb have emerged, such as "A rule of thumb is for the size of this hidden layer to be somewhere between the input layer size and the output layer size" [13] and "How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer" [14]. Rules such as these have little merit since they do not take into account the number of training samples. Others which have an empirical foundation should be treated with caution as they may only work in specific situations or domains, such as [15] and [16].

B. Trial and Error

This is a very unsophisticated approach, unlikely to yield good results except by accident. Occasionally this term is used to mean a bounded exhaustive search, [17] for example, search the space of 9-25 hidden nodes.

C. Exhaustive Search

This involves searching through all the possible topologies within the theoretical bounds and choosing the one with the least generalisation error. The main problem with this approach is it can be very time consuming – especially for networks with two hidden layers where the search complexity is $O(n^2)$. A further problem with this approach is what we call *generalisation jitter* - the generalisation error varies from run to run due to the initial random weight allocation even when all other factors are kept constant. In order to reduce this effect, a number of networks of each topology are trained, the best of which is elected as a candidate. Even then, there can be significant variation in the 'optimum' number of hidden nodes selected even when all else is constant from run to run.

D. Growing Algorithms

These are very similar to exhaustive searches, except that they stop when there is no significant improvement in the generalisation error. This method also suffers from generalisation jitter, which in this case could cause premature stopping due to a local rise in generalisation error. One way to improve on this would be to use a running average of the average generalisation error. However, even this would not help for two dimensional searches due to the sudden variation of the node ratio on a new row. A different

approach might stop when an acceptable level of error is found. However on the one hand, there is no guarantee that it will ever be found, on the other there might be a better solution just round the corner. Other types of growing algorithms combine growing and training simultaneously. These, which can be classified as *evolutionary*, and *non-evolutionary* are beyond the scope of this paper. The former will be investigated in future work, and a survey of the latter for function approximation can be found in [18].

E. Pruning Algorithms

With the pruning approach, a deliberately oversized network is trained, and then the weights are analysed to determine their relative importance. The least important connection weights are removed, and then the network retrained and the whole process repeated. The problem here is in determining *what* constitutes an oversized network in the first instance. Determining which are the least important weights can also be a problem – and this process can be extremely time consuming. A brute force approach which sets each weight to zero and then removes it, if it has a negligible effect on accuracy has a computational complexity of $O(N_s W^3)$, where N_s is the number of samples and W is the number of weights [19]. One of the best known pruning algorithms, *Optimal Brain Surgeon* (OBS) which eliminates weights one by one, takes about 14 hours to prune a SLFN from 5-50-1 to 5-10-1. *Fast Unit Pruning*, based on OBS, is faster as it removes several weights at a time. However, even this takes 5.5 hours to prune the same network [20]. An excellent survey of pruning algorithms can be found in Reed [19].

IV. The Heurix System

Heurix consists of a set of MATLAB modules each with distinct responsibilities. The initialisation phase, shown on the left half of the flow chart in figure 2, prepares the domain data and is only invoked once. The build phase, which is controlled by the 4 parameters on the far right of figure 2, uses the domain data to build the actual neural network. The sampler tests a number of topologies, which the fitter uses to estimate the error curve. From this, the predictor decides on the optimal number of hidden nodes, and finally the scanner selects the champion from a number of candidates with the same topology. Sections A – E describe the modules in greater detail.

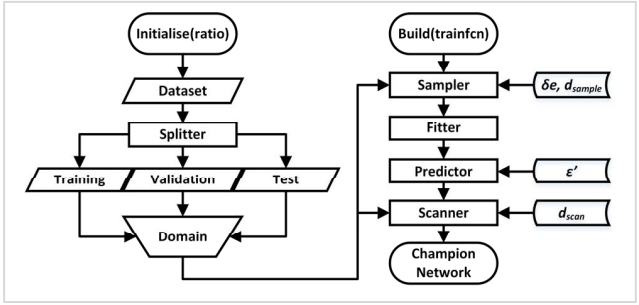

Figure 2. Heurix Flow Chart

A. Splitter Module

This splits the data into training, validation and test sets. Crucially, it ensures that all *keypoint* samples are allocated to the training set. These are the samples which contain the minimum and maximum values of each input and output. This step is important because ANNs are better at interpolating than extrapolating. The remaining samples are randomly split between training, validation and test sets according to the desired split ratio. The training set is used to train the network using the validation set for early stopping, and the test set is used as an estimate of the generalisation error for selecting the champion. The same domain data are used throughout to ensure that the networks are competing on an even playing field.

B. Sampler Module

The Sampler tests a number of representative topologies and passes their errors to the fitter module. Since the error curve is exponential, the most efficient way to do this is to use an exponential scale. The topologies selected are thus the set $N_i - n_k - N_o$, where N_i and N_o are the number of inputs and outputs respectively, k is an integer, and n_k is the number of hidden neurons $n_k = 2^k$, for $0 \leq k \leq k_{ceil}$ The ceiling, k_{ceil} is calculated theoretically from equation (1) as the ceiling of $log_2(N_s)$ Where N_s is the number of training samples. A number of separate networks are trained for each k , recording the average training error ε_k using the normalised root mean square error (NRMSE). The sampler stops early when the error gradient drops below a threshold value δe such that $\varepsilon_k - \varepsilon_{k-1} < \delta e$ per *binade* (defined as the interval between 2^k and 2^{k+1}) AND $k \geq 5$. The quantity k_{max} represents the actual value of k at this point. Thus the output from the sampler module is $n_0, \ldots n_{k_{max}}$ and $\varepsilon_0, \ldots \varepsilon_{k_{max}}$.

C. Fitter Module

The fitter module uses the sampler output to fit an error curve $\varepsilon = f(n)$ of the form:

$$\varepsilon = cn^b + a, \tag{3}$$

where $a, b,$ and c are constants, $0 \leq a < \varepsilon_{min}$, and ε_{min} is the minimum value of $\varepsilon_0 - \varepsilon_{k_{max}}$. This is done iteratively using linear regression of the natural logarithms of ε and n for different values of a. The values of a, b and c which yield the best correlation coefficient $r > 0.99$ are chosen.

D. Predictor Module

This module predicts the optimal number of hidden nodes based on user preference. It does so by calculating the derivative of the error function ε' and solving it to calculate the total number of hidden nodes N_h required for a given rate of change of error per node. These steps are shown in (4) and (5).

$$\varepsilon' = \frac{d\varepsilon}{dn} = bcn^{b-1} \tag{4}$$

$$N_h = \left(\frac{\varepsilon'}{bc}\right)^{\frac{1}{b-1}} \tag{5}$$

The value N_h is passed to the scanner to search for the champion network using a *deep* scan. The choice of ε' detemines the balance between network complexity and network accuracy. Three default values of ε' determine whether the network will be optimised for *speed* (low complexity), *accuracy* or *balanced* between the two.

E. Scanner

The scanner has two basic modes of operation, *wide* and *deep*. In wide mode it performs an exhaustive search of topologies between two limits through one dimensional space. In deep mode it focuses on a single topology with N_h hidden nodes. A number of networks, d_{scan}, are created for each topology. It returns the overall champion, which is the network with the lowest error on the unseen test data set – i.e. the best generaliser.

V. Experiments and Results

All experiments were carried out using the MATLAB R2014b neural network toolbox on an Acer Aspire V 17 Nitro (Black Edition) laptop. This is fitted with a core i7 4710HQ processor running at 2.5GHz.

The MATLAB Engine Data Set was used to develop the algorithm and run all the experiments. This contains 1199 samples organised as two inputs (fuel and speed) and two outputs (torque and NO_x). Prior to being fed to the splitter, these were reorganised into 3 inputs (torque, fuel and speed) and a single NO_x output. These were then apportioned by the splitter module into 80% training and 10% each for validation and test (959, 120, and 120 respectively). The same data was used for all of the experiments. In all cases, the training algorithm was 'trainlm' with default parameters - notably 1000 epochs, 0 performance goal, and 6 validation failures. Network Configuration: processFcn = *'mapminmax'*, divideFcn = *'divideind'*, performFcn = *'mse'*, transferFcn = *'tansig'*. In all cases, the normalised root mean square (NRMSE) is used as a measure of the error:

$$NRMSE_y = \frac{1}{\hat{y}_{max} - \hat{y}_{min}} \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{N_d}}, \qquad (6)$$

Where N_d represents the number of data points in the data set, $\hat{y}_i$ is the expected value, and y_i is the actual value.

Following an investigative phase which assessed the accuracy and repeatability of results as well as general sensitivity to the settings, the defaults shown in table 1 were selected.

The heart of the Heurix system is the curve fitter module, from which the optimum number of nodes is predicted. Its accuracy and repeatability are tested by comparing its output over 10 runs to the actual data from a full scan. Figure 3 shows the average training error over 30 networks for each topology between 1 and 256 hidden nodes, overlaid on one such fitter output. The root mean square error (rmse) between the fitter output and the actual data as well as the correlation coefficient r between the two is shown in table II. There is a remarkably good correlation between the fitter and actual data.

TABLE I. Heurix Default Settings

Parameter	Description	Value / Meaning
d_{sample}	Sampler Depth	30 networks
δe	Sampler Stop Gradient	-1.00% per binade
d_{scan}	Scanner Depth	30 networks
speed	Optimisation setting	ε'= -0.10% per node
balanced	Optimisation setting	ε'= -0.05% per node
accuracy	Optimisation setting	ε'= -0.02% per node

TABLE II. Average Fitter Correlation

	rmse	r
μ	0.0012	0.9953
σ	0.00023	0.00041

TABLE III. Average Performance Summary

Optimisation Mode	N_h	w	Overall Error %	Generalisation Error %	Run Time (min)
exhaustive	55.5	278.5	0.85	0.87	112.1
accuracy	35.8	180.0	1.04	1.27	7.9
balanced	20.6	104.0	1.44	1.74	7.4
speed	13.3	67.5	1.95	2.28	5.5

The performance of Heurix is compared with exhaustive search by selecting the champion generaliser from 10 runs of each optimisation setting. The average number of hidden nodes N_h, weights w, overall error, generalisation error and run time are shown in table III. The weights (including biases) are calculated as: $w = \sum_{i=1}^{l} (n_{i-1} + 1) \times n_i$, where l is the number of layers (including input and output layers); n_0 is the number of inputs, n_i is the number of nodes in layer i and n_l is the number of outputs. These are used to estimate relative network response times. Figure 4 shows the overall and generalisation errors. The latter are just +0.4%, +0.9% and +1.4% relative to exhaustive search. In figure 5, the build time and network response time are shown as a percentage of the time for an exhaustive search. There is an insignificant difference in build time between the modes - they all build networks in between 5%-7% of the time of an exhaustive search. In terms of the response time, estimated from the weights, networks respond on average in 64.6%, 37.3% and 24.2% of the time of those found by exhaustive search. The network build and response of a balanced network is shown in figures 6 and 7. It has a generalisation error of just 1.5% and correlation $r > 0.998$ for all data sets. It took just 6.6 minutes to build.

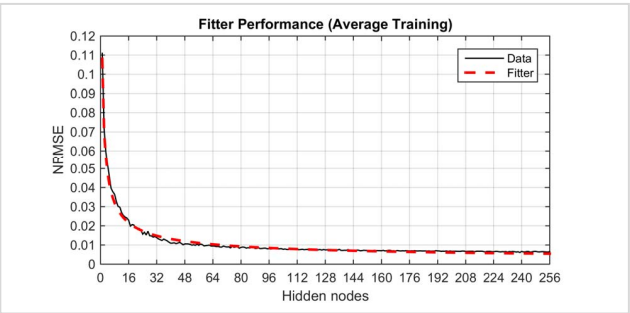

Figure 3. Fitted Curve Overlaid on Actual Training Error

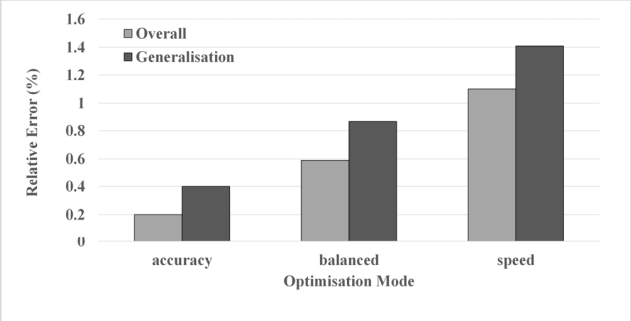

Figure 4. Relative Errors

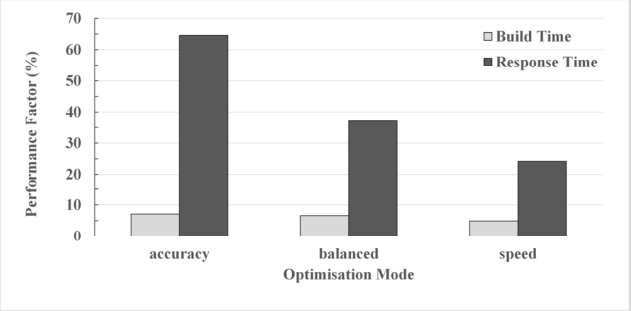

Figure 5. Relative Performance

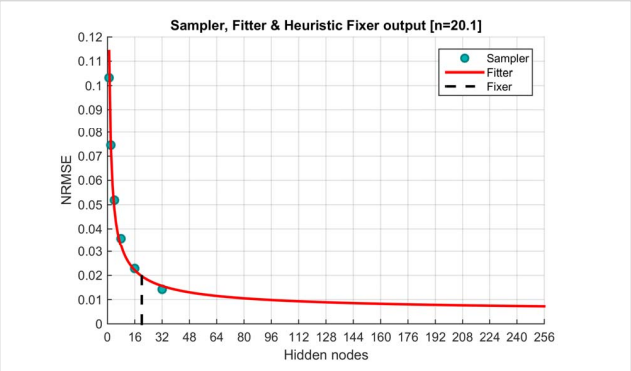

Figure 6. Network Build

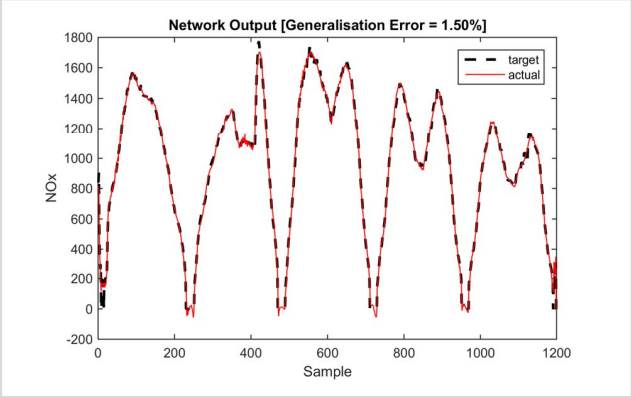

Figure 7. Network Response

VI. Conclusions

In summary, there are still no mathematically founded theoretical methods of predicting the number of hidden nodes. It is argued that practical methods of fixing these must necessarily be done *in-situ* using the actual domain data, and training algorithm to be deployed. Current methods which do so, such as exhaustive search, growing, pruning and evolutionary algorithms are extremely time consuming. In order to offer a faster alternative, a novel system called Heurix was developed and its performance compared to exhaustive search (ES). On average Heurix can build networks in 5-7% of the time of ES (<8mins). These have generalisation errors between 0.4-1.4% greater than ES, and response time 24-65% of ES, depending on which optimisation setting is used. Heurix is showing great promise but needs testing with other datasets.

Glossary of Terms

ANN – Artificial Neural Network
LM – Levenberg Marquardt training Algorithm
NRMSE – Normalised Root Mean Square Error
OBS – Optimal Brain Surgeon.
SLFN – Single hidden Layer Feed-forward Network
TLFN – Two hidden Layer Feed-forward Network
N_h - Total number of hidden nodes or neurons
N_s – Number of training samples
N_o – Number of outputs
N_i – Number of inputs
δe – Sampler stop gradient
ε' – Fixer gradient rate of change of error per node.

Acknowledgments

We thank Prof. Martin T. Hagan of Oklahoma State University for kindly donating the Engine Data Set used in this paper to Matlab. We also extend our gratitude to the anonymous reviewers for their much valued positive feedback and suggestions. Last but not least, A.J.T. thanks Mavis for her infinite tolerance.

References

[1] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 224–229, Jan. 1998.

[2] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 274–281, Mar. 2003.

[3] P. L. Bartlett, "For valid generalization, the size of the weights is more important than the size," *Adv. Neural Inf. Process. Syst.*, vol. 9, p. 134, 1997.

[4] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Dokl. Akad. Nauk USSR*, vol. 114, pp. 953–956, 1957.

[5] G. G. Lorentz, "Approximation of functions. 1966," *Rinehart Winst. N. Y.*

[6] D. A. Sprecher, "On the structure of continuous functions of several variables," *Trans. Am. Math. Soc.*, vol. 115, pp. 340–355, 1965.

[7] R. Hecht-Nielsen, “Kolmogorov’s mapping neural network existence theorem,” in *Proceedings of the international conference on Neural Networks*, 1987, vol. 3, pp. 11–14.

[8] K. Swingler, *Applying Neural Networks: A Practical Guide*. Morgan Kaufmann, 1996.

[9] R. Hecht-Nielsen, *Neurocomputing*, Reprint. Boston, Ma.: Addison-Wesley, 1991.

[10] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[11] V. Kůrková, “Kolmogorov’s theorem and multilayer neural networks,” *Neural Netw.*, vol. 5, no. 3, pp. 501–506, 1992.

[12] S. Tamura and M. Tateishi, “Capabilities of a four-layered feedforward neural network: four layers versus three,” *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 251–255, Mar. 1997.

[13] A. Blum, *Neural Networks in C++: An Object-oriented Framework for Building Connectionist Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1992.

[14] M. J. Berry and G. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Support*. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[15] J. Ke and X. Liu, “Empirical Analysis of Optimal Hidden Neurons in Neural Network Modeling for Stock Prediction,” in *Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA ’08*, 2008, vol. 2, pp. 828–832.

[16] K. Shibata and Y. Ikeda, “Effect of number of hidden neurons on learning in large-scale layered neural networks,” in *ICCAS-SICE, 2009*, 2009, pp. 5008–5013.

[17] J. Li, C. He, and D. Jia, “Emission Modeling of Diesel Engine Fueled with Biodiesel Based on Back Propagation Neural Network,” in *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, 2010, vol. 4, pp. 379 –381.

[18] T.-Y. Kwok and D.-Y. Yeung, “Constructive algorithms for structure learning in feedforward neural networks for regression problems,” *Neural Netw. IEEE Trans. On*, vol. 8, no. 3, pp. 630–645, 1997.

[19] R. Reed, “Pruning algorithms-a survey,” *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, 1993.

[20] J. Qiao, Y. Zhang, and H. Han, “Fast unit pruning algorithm for feedforward neural network design,” *Appl. Math. Comput.*, vol. 205, no. 2, pp. 622–627, Nov. 2008.