Constructive Neural Networks: Some Practical Considerations

Tin-Yau Kwok

Dit-Yan Yeung

Abstract—Based on a Hilbert space point of view, we proposed in our previous work a novel objective function for training new hidden units in a constructive feedforward neural network. Moreover, we proved that if the hidden unit functions satisfy the universal approximation property, the network so constructed incrementally, using the proposed objective function and with input weight freezing, still preserves the universal approximation property with respect to L^2 performance criteria. In this paper, we provide experimental support for the feasibility of using this objective function. Experiments are performed on two chaotic time series with encouraging results. In passing, we also demonstrate that engineering problems are not to be neglected in practical implementations. We identify the problem of plateau, and then show that by suitably transforming the objective function and modifying the quickprop algorithm, significant improvement can be obtained.

I. INTRODUCTION

The approximation capabilities of feedforward neural networks have been investigated by several researchers [4, 7, 8, 9]. However, these results assert no theoretical bounds on the number of hidden units required.

In recent years, attempts have been made to determine the number of hidden units in an automatic way [1, 6, 17]. A particularly successful algorithm is the cascade-correlation architecture [6]. It begins with a minimal network with no hidden units, then automatically trains and adds new hidden units one at a time to create a multi-layer network. The hidden units are added in a greedy manner. A desirable new hidden unit maximizes the following objective function

$$S_{cascor} = \sum_o \left| \sum_p (E_{po} - \bar{E}_o)(H_p - \bar{H}) \right|,$$

The authors are with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. This research has been partially supported by the Hong Kong Research Grants Council and the Hong Kong Telecom Institute of Information Technology. The first author is also supported by the Sir Edward Youde Memorial Fellowship.

where o ranges over the output units, p ranges over the training patterns, H_p is the activation of the new hidden unit for pattern p , E_{po} is the residual error of output unit o for pattern p before the new hidden unit is added, and $\bar{H}$ and $\bar{E}_o$ are the corresponding average values over all patterns. After the new unit is added, all the weights between the input and hidden units as well as those among the hidden units are held constant, allowing only the hidden-to-output weights to change. This heuristic was referred to as *input weight freezing*. By employing this heuristic and the cascade-correlation architecture together, Fahlman and Lebiere obtained fast learning under all the cases tested.

However, the design of the function S_{cascor} is rather *ad hoc*. Moreover, although references like [8] proved that neural networks can be universal approximators, it is unclear whether this property can still be preserved when input weight freezing is used.

In [12], we formulated the problem of learning in constructive neural networks as constructive approximation in a Hilbert space. Using a greedy approach, we proposed the following objective function, S , for training new hidden units in networks with linear output units:

$$S = \frac{\sum_o (\sum_p E_{po} H_p)^2}{\sum_p H_p^2}. \quad (1)$$

The resultant network construction algorithm is similar to the cascade-correlation architecture. Moreover, we proved that if the hidden unit functions satisfy the universal approximation property, the network so constructed incrementally, with input weight freezing, still preserves the universal approximation property with respect to L^2 performance criteria.

This paper presents experimental results in using the proposed objective function (1) for learning chaotic time series. The network so constructed has a single hidden layer, as in [17], with sigmoidal hidden units and linear output units. Section II describes problems in practical optimization of S . Section III describes two time series that are used in this experiment, with simulation results in section

IV. The last section gives some concluding remarks.

II. PRACTICAL PROBLEMS

The network construction algorithm described in [12] consists of two phases. The first phase selects the new hidden unit that is to be installed in the network, by maximizing S with respect to the weights that are connected from all input units to the new hidden unit. The second phase adjusts all the hidden-to-output weights after the hidden unit is installed. Because the output units are linear, this phase may be done by computing the pseudo-inverse exactly. The first phase, however, is a nonlinear optimization problem, and is thus more problematic.

This optimization process may end up in local optima. Although [12] showed that the universal approximation property of the resultant network is not affected by these local optima, in real applications we may still want to have local optima that are not too bad. The reason is that we want to have fewer hidden units in the network; too many hidden units may degrade the generalization performance.

A. Problem with the Objective Function

Denoting the parameters associated with the candidate hidden unit by Π , and differentiating S in (1) with respect to Π , we obtain

$$\begin{aligned} \nabla_{\Pi} S = & \frac{2}{(\sum_p H_p^2)^2} \sum_o \{ (\sum_p E_{po} H_p) [(\sum_p H_p^2) \\ & (\sum_p E_{po} \nabla_{\Pi} H_p) - (\sum_p E_{po} H_p) (\sum_p H_p \nabla_{\Pi} H_p)] \}, \end{aligned}$$

and similarly for $\nabla_{\Pi}^2 S$.

As learning proceeds, hidden units are added to the network, and the residual errors E_{po} 's decrease with time. Observe that S , $\nabla_{\Pi} S$ and $\nabla_{\Pi}^2 S$ are continuous with respect to E_{po} , and

$$S = \nabla_{\Pi} S = \nabla_{\Pi}^2 S = 0,$$

when all the E_{po} 's are zero. Hence,

$$S \rightarrow 0, \quad \nabla_{\Pi} S \rightarrow 0, \quad \nabla_{\Pi}^2 S \rightarrow 0,$$

as $E_{po} \rightarrow 0 \quad \forall p \forall o$. This may pose a problem when the residual errors are small but still not acceptable. If gradient-ascent algorithms that use only first-order information (such as standard back-propagation) are used in the optimization, Π is changed at each step by an amount

$$\Delta \Pi = \eta \nabla_{\Pi} S, \quad (2)$$

where η is the learning rate. As $\nabla_{\Pi} S$ is small, learning basically comes to a halt. By using ascent algorithms that use second-order information, we have

a better hope of further maximizing S . Hence, in this experiment, quickprop [5] is used to implement this optimization. However, the problem is still not solved completely.

B. Problem with Quickprop

The quickprop algorithm [5] is a second-order method. It assumes that the function, S , to be optimized is locally quadratic with respect to each weight, and the Hessian matrix of S in weight space is diagonal. Although these assumptions are quite "risky", the technique works well in practice [5, 11].

Denoting $\frac{\partial S}{\partial w_i}$ by S'_i , the change for weight w_i at time t is given by:

$$\Delta w_i[t] = \frac{S'_i[t]}{S'_i[t-1] - S'_i[t]} \Delta w_i[t-1], \quad (3)$$

where $\Delta w_i[t-1]$ is the weight update at time $t-1$, while $S'_i[t]$ and $S'_i[t-1]$ are the derivatives at times t and $t-1$ respectively.

There are cases when weight update by (3) is not used. For example, since quickprop changes weights based on what happened during the previous weight update, (2) is used to compute the first step.

Another situation is when the current slope with respect to w_i is in the same direction as that of the previous slope, and the magnitude is decreasing but still comparable, i.e.

$$S'_i[t] S'_i[t-1] > 0, \quad (4)$$

$$|S'_i[t-1]| > |S'_i[t]| > \lambda |S'_i[t-1]|, \quad \lambda < 1. \quad (5)$$

In this case, the next weight step is given by

$$\Delta \Pi[t] = \mu \Delta \Pi[t-1], \quad \mu > 1. \quad (6)$$

Although the aim of this restriction is to improve the stability of the algorithm, it may be problematic when one is exploring in a plateau¹ in the weight space. Consider the w_i direction in the weight space. Taking the gradient ascent step in (2) at $t=0$,

$$w_i[1] = w_i[0] + \eta S'_i[0],$$

with

$$\Delta w_i[0] = \eta S'_i[0]. \quad (7)$$

Now, using Taylor series expansion,

$$\begin{aligned} \frac{S'_i[1]}{S'_i[0]} & \simeq \frac{S'_i[0] + \eta S'_i[0] S''_i[0]}{S'_i[0]} \\ & = 1 + \eta S''_i[0], \end{aligned} \quad (8)$$

¹We define a *plateau* as a region where the first and second derivatives of the function to be optimized with respect to all the parameters are nearly zero.

where S_i'' denotes $\frac{\partial^2 S}{\partial w_i^2}$. If $w_i[0]$ falls in the region of a plateau, then

$$S_i'[0] \simeq 0, \quad S_i''[0] \simeq 0.$$

(8) then implies

$$\frac{S_i'[1]}{S_i'[0]} \simeq 1,$$

which satisfies conditions (4) and (5). The next weight step, using (6) and (7), is

$$\Delta w_i[1] = \mu \eta S_i'[0] \simeq 0, \quad (9)$$

movement in the weight space is thus very slow.

This problem, however, is usually not that severe. Although the next weight step is small, and even if conditions (4) and (5) continue to be satisfied, it can build up gradually given a sufficient number of training epochs. This can be seen clearly by applying (6) repeatedly,

$$\Delta \Pi[t] = \mu^n \Delta \Pi[t - n].$$

Thus, the difference between $S_i'[t]$ and $S_i'[t - 1]$ will finally be large enough for the quadratic approximation to be applied.

However, in constructive neural network algorithms such as [1, 6, 12], training is usually limited by a *patience* parameter, which means that the function to be optimized has to improve by a certain fraction within a certain number of training epochs. Patience helps to end each learning phase when progress is excessively slow, thus it saves time and is also experimentally shown to be able to improve generalization [16]. However, the value of this patience parameter is usually small. Hence, waiting for the weight slopes to slowly build up is not practical under such situations, and more drastic action is needed to get out of the plateau in limited time. This situation is particularly acute in our case, as we have shown in the last section that when the residual errors are small, both the first and second derivatives of the objective function are likely to be small.

C. Remedies

To alleviate the problems mentioned above, we aim at increasing the slope of the objective function to be optimized when the residual errors are small, such that the region to be searched is less likely to be a plateau. On the other hand, if we are so unfortunate that quickprop really searches in a plateau, we aim at finding a better method to get out of it quickly.

To avoid the first problem, Fahlman [5] suggested to adjust $\nabla_{\Pi} S$ by adding a small offset to the values of $\nabla_{\Pi} H_p$. However, this distorts the true value

of $\nabla_{\Pi} S$, and as noted by Crowder [2] and verified experimentally by some of our preliminary results, this “confuses the correlation machinery” [2], and cannot solve the problem.

C.1. Transforming the Objective Function

To increase the slope of S when the residual errors are small, we transform S by a (possibly nonlinear) functional $f : \mathcal{C} \rightarrow \mathcal{C}$ (where $\mathcal{C}$ is the space of all real-valued continuous functions),

$$\tilde{S} = f(S),$$

such that

$$\tilde{S}_i' \geq S_i'$$

when S is small. Moreover,

$$(f(S))(\{E_{po}\}, \{H_p\}) = \hat{f}(S(\{E_{po}\}, \{H_p\})),$$

where $\hat{f} : \mathfrak{R} \rightarrow \mathfrak{R}$ is strictly increasing. This is required so that locations of the relative and global optima of $\tilde{S}$ are the same as those of S .

An obvious choice of f is

$$f(S) = aS, \quad a > 1.$$

This amounts to scaling up the S dimension, or equivalently increasing the learning rate η in (2). However, increasing η too much may cause oscillation, while increasing only a little may not be useful in improving the situation.

Another simple choice is

$$f(S) = \sqrt{S} = \frac{|\sum_p E_p H_p|}{\sqrt{\sum_p H_p^2}}. \quad (10)$$

Now

$$\begin{aligned} \tilde{S}_i' &= \frac{1}{2\sqrt{S}} S_i', \\ \tilde{S}_i' > S_i' &\Leftrightarrow S < 0.25. \end{aligned} \quad (11)$$

Thus, the slope is scaled up when S is small, and the smaller the S , the larger the scaling factor (figure 1). Experimentally, (11) almost always holds (except for the first few hidden units). Hence, the resulting $\tilde{S}$ surface is usually steeper than that of S , making the region to be searched less likely to be a plateau. Even when S is large, which makes $\tilde{S}_i'$ smaller than S_i' , it is not a problem because then quickprop will be mainly using the quadratic approximation (3).

Of course, there are other choices of f , such as

$$f(S) = S^{\frac{1}{n}}, \quad n > 1,$$

which basically changes the switch-over point where $\tilde{S}_i' > S_i'$. But the basic idea of all these schemes is

to dynamically alter the slope of the objective function during learning. This is similar to the idea of adaptive learning rate in back-propagation [10, 11]. However, we demonstrate here that a simple change in the objective function to be optimized can achieve the same goal, without requiring modification to the learning algorithm or continual updating of the learning rate which incurs additional computational burden.

Figure 1: Plot of several choices of f .

C.2. Modification to Quickprop

For the second problem of enabling a faster escape from the plateau, a simple solution is to take large steps [15]. From section II.B, we saw that the problem arises from always using the gradient ascent step when conditions (4) and (5) are satisfied. To alleviate this, we take the quadratic approximation in (3) when the changes in all weight directions are very small, even under those conditions.

Following the analysis in section II.B, by taking the quadratic ascent step,

$$\begin{aligned}\Delta w_i[1] &= \frac{S'_i[1]}{S'_i[0] - S'_i[1]} \Delta w_i[0] \\ &\simeq \frac{\eta S'_i[0](S'_i[0] + \eta S'_i[0] S''_i[0])}{-\eta S'_i[0] S''_i[0]} \\ &= -S'_i[0] \left(\eta + \frac{1}{S''_i[0]} \right).\end{aligned}$$

This can be shown to be greater than the gradient ascent step in (9) if

$$|S''_i[0]| < \frac{1}{\eta(1 + \mu)},$$

which is usually satisfied when $w_i[0]$ falls on a plateau.

III. BENCHMARK PROBLEMS

Simulation is performed on two common benchmarks. The first one is the logistic series, defined by

$$x[t+1] = \lambda x[t](1 - x[t]). \quad (12)$$

When $\lambda = 4$, the iterates of equation (12) form a chaotic series [13]. The goal is to predict the value $x[t+1]$ given the present value $x[t]$.

The second problem is the less trivial Mackey-Glass series, derived by integrating the equation:

$$\dot{x}[t] = \frac{ax[t - \tau]}{1 + x[t - \tau]^{10}} - bx[t].$$

When $a = 0.2$, $b = 0.1$, and $\tau = 17$, the integration produces a chaotic time series [13]. Following the standard approach [3, 13, 14], each training sample contains the points $x[t-18]$, $x[t-12]$, $x[t-6]$ and $x[t]$. Prediction is then made to the future point $x[t+6]$. By feeding this predicted value back into the input and iterating the solution, prediction at $P = 84$ is also obtained.

IV. SIMULATION

For the logistic series, the numbers of training and testing samples are both 200. The network is trained until 30 hidden units are installed, and repeated for 30 trials. For the Mackey-Glass series, the numbers of training and testing samples are both 500, with a maximum of 100 hidden units. 100 trials are performed.

Both RMS error and error index² (quoted in brackets) are reported. These are taken when the maximum allowable number of hidden units is installed, and averaged over all trials. As over-training may have occurred, the average of the best generalization performance for each trial is also reported. Besides, Student's t -test is performed on the results to assure that the differences are statistically significant with a confidence level of at least 95%.

Tables 1 and 2 compare the results of using S in (1) versus $\tilde{S}$ in (10) as the objective function, using the modified quickprop algorithm. Tables 3 and 4 compare the results of using the original quickprop algorithm versus the modified algorithm, with $\tilde{S}$ as the objective function. Table 5 compares the results of using S with the original quickprop versus $\tilde{S}$ with the modified quickprop. Clearly both the training and testing performances get improved when using $\tilde{S}$ and the modified algorithm.

Using the combination of $\tilde{S}$ as the objective function with the modified quickprop algorithm, figure 2 compares the predicted values from the network with the desired values for the logistic series. Figure 3 compares the predicted values with the desired values for the Mackey-Glass series at $P = 84$.

²Error index is defined as the RMS error divided by the standard deviation of the series [13].

The curves in each plot basically overlap, illustrating that the network has learnt the mapping accurately.

Table 6 compares our results with the cascade-correlation architecture in learning the Mackey-Glass series [3]. Our proposed method is able to further decrease the residual error, and the performance is comparable with techniques like traditional back-propagation, linear predictive method and multi-resolution hierarchies as reported in [3].

V. CONCLUSION

In [12], we proved that if the hidden unit functions satisfy the universal approximation property, the network so constructed incrementally, using the proposed objective function (1) and with input weight freezing, still preserves the universal approximation property with respect to L^2 performance criteria. In this paper, we provide experimental support for the feasibility of using this objective function. It even outperforms the cascade-correlation architecture in predicting the Mackey-Glass time series.

In passing, we also demonstrate that engineering problems are not to be neglected in practical implementations. A simple transformation of the objective function to be optimized, although theoretically identical to the original, may lead to significantly different results in the numerical optimization process. Moreover, we also improve quickprop in the handling of plateau, which is especially important for constructive neural network algorithms using the patience parameter.

Figure 2: Predicted and desired values for the logistic series.

REFERENCES

- [1] T. Ash. Dynamic node creation in backpropagation networks. ICS Report 8901, Institute for Cognitive Science, University of California, San Diego, 1989.
- [2] R.S. Crowder. Cascor.c, C implementation of the cascade-correlation learning algorithm, 1990.
- [3] R.S. Crowder. Predicting the Mackey-Glass timeseries with cascade-correlation learning. In *Proceedings of the*

Figure 3: Predicted and desired values for the Mackey-Glass series at $P = 84$.

- Connectionist Models Summer School*, pages 117–123, 1990.
- [4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
- [5] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, 1988.
- [6] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, 1990.
- [7] K.I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.
- [8] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.
- [9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [10] R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.
- [11] T.T. Jarvis and W.J. Fitzgerald. Optimization schemes for neural networks. Technical Report TR 144, Cambridge University Engineering Department, 1993.
- [12] T.Y. Kwok and D.Y. Yeung. Theoretical analysis of constructive neural networks. Technical Report HKUST-CS93-12, Department of Computer Science, Hong Kong University of Science and Technology, 1993. Submitted to *Neural Computation*.
- [13] A. Lapedes and F. Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. LA-UR 87-2662, Los Alamos National Laboratory, 1987.
- [14] J. Moody and C. Darken. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143, 1988.
- [15] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, 1993.
- [16] C.S. Squires and J.W. Shavlik. Experimental analysis of aspects of the cascade-correlation learning architecture. Machine Learning Research Group Working Paper 91-1, Computer Sciences Department, University of Wisconsin-Madison, 1991.
- [17] D.Y. Yeung. Constructive neural networks as estimators of Bayesian discriminant functions. *Pattern Recognition*, 26(1):189–204, 1993.

Table 1: Comparison of S versus $\tilde{S}$ for the logistic series.

	S	$\tilde{S}$	improvement
training (average)	0.00350(0.0099)	0.00234(0.0066)	↓ 33.1%
testing (average)	0.00390(0.011)	0.00273(0.0077)	↓ 30.1%
testing (best)	0.00383(0.011)	0.00271(0.0076)	↓ 29.3%

Table 2: Comparison of S versus $\tilde{S}$ for the Mackey-Glass series.

	S	$\tilde{S}$	improvement
training (average)	0.00624(0.0259)	0.00604(0.0250)	↓ 3.3%
testing for P=6 (average)	0.00789(0.0327)	0.00768(0.0318)	↓ 2.7%
testing for P=6 (best)	0.00787(0.0326)	0.00765(0.0317)	↓ 2.8%
testing for P=84 (average)	0.0287(0.119)	0.0282(0.117)	statistically insignificant
testing for P=84 (best)	0.0270(0.112)	0.0261(0.108)	statistically insignificant

Table 3: Comparison of the original versus modified quickprop algorithms using $\tilde{S}$ for the logistic series.

	original	modified	improvement
training (average)	0.00366(0.010)	0.00234(0.0066)	↓ 36.1%
testing (average)	0.00411(0.012)	0.00273(0.0077)	↓ 33.6%
testing (best)	0.00406(0.011)	0.00271(0.0076)	↓ 33.4%

Table 4: Comparison of the original versus modified quickprop algorithms using $\tilde{S}$ for the Mackey-Glass series.

	original	modified	improvement
training (average)	0.00708(0.0293)	0.00604(0.0250)	↓ 14.8%
testing for P=6 (average)	0.00860(0.0356)	0.00768(0.0318)	↓ 10.7%
testing for P=6 (best)	0.00858(0.0356)	0.00765(0.0317)	↓ 10.9%
testing for P=84 (average)	0.0318(0.132)	0.0282(0.117)	↓ 11.2%
testing for P=84 (best)	0.0295(0.122)	0.0261(0.108)	↓ 11.6%

Table 5: Comparison of S with the original quickprop versus $\tilde{S}$ with the modified quickprop for the Mackey-Glass series.

	S with original quickprop	$\tilde{S}$ with modified quickprop	improvement
training (average)	0.0149(0.0616)	0.00604(0.0250)	↓ 59.4%
testing for P=6 (average)	0.0165(0.0684)	0.00768(0.0318)	↓ 53.5%
testing for P=6 (best)	0.0165(0.0682)	0.00765(0.0317)	↓ 53.6%
testing for P=84 (average)	0.0583(0.242)	0.0282(0.117)	↓ 51.6%
testing for P=84 (best)	0.0536(0.222)	0.0261(0.108)	↓ 51.3%

Table 6: Error index comparison with the cascade-correlation architecture for the Mackey-Glass series.

	cascade-correlation	proposed	improvement
testing for P=6 (average)	0.06	0.0318	↓ 47.0%
testing for P=6 (best among all trials)	0.04	0.0241	↓ 39.8%
testing for P=84 (average)	0.32	0.117	↓ 63.4%
testing for P=84 (best among all trials)	0.17	0.0745	↓ 56.2%