Appeared in: Neural Computation, vol. 3, no. 2, pp. 213–225, (Summer 1991).

A Resource-Allocating Network for Function Interpolation

John Platt
Synaptics
2860 Zanker Road, Suite 206
San Jose, CA 95134

We have created a network that allocates a new computational unit whenever an unusual pattern is presented to the network. This network forms compact representations, yet learns easily and rapidly. The network can be used at any time in the learning process and the learning patterns do not have to be repeated. The units in this network respond to only a local region of the space of input values.

The network learns by allocating new units and adjusting the parameters of existing units. If the network performs poorly on a presented pattern, then a new unit is allocated which corrects the response to the presented pattern. If the network performs well on a presented pattern, then the network parameters are updated using standard LMS gradient descent.

We have obtained good results with our resource-allocating network (RAN). For predicting the Mackey Glass chaotic time series, our network learns much faster than do those using back-propagation and uses a comparable number of synapses.

1 Introduction

(Judd, 1988) has shown that the problem of loading a multilayer perceptron with binary units is NP-complete. Loading sigmoidal multilayer networks is computationally expensive for large sets of real data, with unknown bounds on the amount of computation required.

(Baum, 1989) pointed out that the problem of NP-complete loading is associated only with a network of fixed resources. If a network can allocate new resources, then the problem of loading can be solved in polynomial time. Therefore, we are interested in creating a network that allocates new computational units as more patterns are learned.

Traditional pattern recognition algorithms, such as Parzen windows and k-nearest neighbors, allocate a new unit for every learned example. The number of examples in real problems forces us to use fewer than one unit for every learning example: we must create and store an abstraction of the data.

The network described here allocates far fewer units than the number of presented examples. The number of allocated units scales sub-linearly with the number of presented inputs. The network can be used either for on-line or off-line learning.

Previous workers have used networks whose transfer function is a Gaussian (Broomhead & Lowe, 1988) (Moody & Darken, 1988 & 89) (Poggio & Girosi, 1990). The use of Gaussian units were originally inspired by approximation theory, which describes algorithms that interpolate between irregularly spaced input-output pairs (Powell, 1987). In fact, Lapedes discussed the hypothesis that multiple layers of sigmoidal units form Gaussian-like transfer functions in order to perform interpolation (Lapedes, 1987).

Gaussian units are well-suited for use in a resource-allocating network because they only respond to a local region of the space of input values. When a Gaussian unit is allocated, it explicitly stores information from an input-output pair instead of merely using that information for gradient descent. The explicit storage of an input-output pair means that this pair can be used immediately to improve the performance of the system in a local region of the input space. A unit with a non-local response needs to undergo gradient descent, because it has a non-zero output for a large fraction of the training data.

Moody and Darken's work (Moody & Darken, 1988 & 89) is the closest to the work specified below. They use Gaussian units, where the Gaussians have variable height, variable centers, and fixed widths. The network learns the centers of the Gaussians using the K-means algorithm (Lloyd, 1957) (Stark, et. al, 1962) (MacQueen, 1967), and learns the heights of the Gaussians using the LMS gradient descent rule (Widrow,

1960). The width of the Gaussians is determined by the distance to the nearest Gaussian center after the K-means learning.

Moody has further extended his work by incorporating a hash table lookup (Moody, 1989). The hash table is a resource-allocating network where the values in the hash table only become non-zero if the entry in the hash table is activated by the corresponding presence of non-zero input probability.

Our work improves on previous work in several ways:

1. Although it has the same accuracy, our network requires fewer weights than do networks in either (Moody and Darken, 1989) or in (Moody, 1989).
2. Like the hashing approach in (Moody, 1989), our network automatically adjusts the number of units to reflect the complexity of the function that is being interpolated. Fixed-size networks either use too few units, in which case the network memorizes poorly, or too many, in which case the network generalizes poorly.
3. We use units that respond to only a local region of input space, similar to (Moody & Darken, 1988 & 89), but unlike back-propagation. The units respond to only a small region of the space of inputs so that newly allocated units do not interfere with previously allocated units.
4. The RAN adjusts the centers of the Gaussian units based on the error at the output, like (Poggio & Girosi, 1990). Networks with centers placed on a high-dimensional grid, such as (Broomhead & Lowe, 1988) and (Moody, 1989), or networks that use unsupervised clustering for center placement, such as (Moody & Darken, 1988 & 89) generate larger networks than RAN, because they cannot move the centers to increase the accuracy.
5. Parzen windows and K-nearest neighbors both require a number of stored patterns that grow linearly with the number of presented patterns. With our method, the number of stored patterns grows sublinearly, and eventually reaches a maximum.

2 The Algorithm

This section describes a resource-allocating network (RAN), which consists of a network, a strategy for allocating new units, and a learning rule for refining the network.

2.1 The Network The RAN is a two-layer network (Figure 1). The first layer consists of units that respond to only a local region of the space of input values. The second layer aggregates outputs from these units and creates the function that approximates the input-output mapping over the entire space.

The units on the first layer store a particular region in the input space. When the input moves away from the stored region the response of the unit decreases. A simple function that implements a locally tuned unit is a Gaussian:

$$\begin{aligned} z_j &= \sum_k (c_{jk} - I_k)^2, \\ x_j &= \exp(-z_j / w_j^2). \end{aligned} \tag{2.1}$$

We use a C^1 continuous polynomial approximation to speed up the algorithm, without loss of network accuracy:

$$x_j = \begin{cases} \left(1 - (z_j / q w_j^2)\right)^2, & \text{if } z_j < q w_j^2; \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

where $q = 2.67$ is chosen empirically to make the best fit to a Gaussian.

The input to the synapses of the second layer are the outputs of the units of the first layer. The purpose of each second-layer synapse is to define the contribution of each first-layer unit to a particular output $\vec{y}$ of the network. Each output of the network $\vec{y}$ is the sum of the first-layer outputs x_j, each weighted by the synaptic strength $\vec{h}_j$ plus a constant vector $\vec{\gamma}$, which does not depend on the output of the first layer:

$$\vec{y} = \sum_j \vec{h}_j x_j + \vec{\gamma}. \tag{2.3}$$

The $\vec{\gamma}$ is the default output of the network when none of the first-layer units are active. The $\vec{h}_j x_j$ term can be thought of as a bump that is added or subtracted to the constant term $\vec{\gamma}$ to yield the desired function.

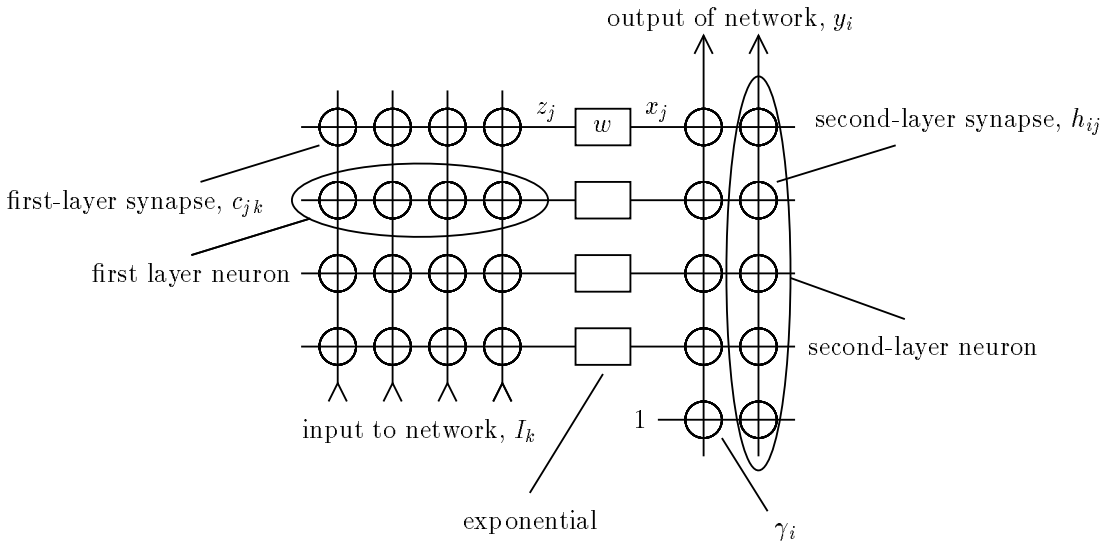

Figure 1: The architecture of the network. In parallel, the network computes the distances of the input vector $\vec{I}$ to the stored centers $\vec{c}_j$. The distance is then exponentiated to yield a weight x_j. The output $\vec{y}$ is a weighted sum of the heights $\vec{h}_j$ and an offset $\vec{\gamma}$.

2.2 The Learning Algorithm The network starts with a blank slate: no patterns are yet stored. As patterns are presented to it, the network chooses to store some of them. At any given point the network has a current state, which reflects the patterns that have been stored previously.

The allocator identifies a pattern that is not currently well represented by the network and allocates a new unit that memorizes the pattern. The output of the new unit extends to the second layer. After the new unit is allocated, the network output is equal to the desired output $\vec{T}$. Let the index of this new unit be n.

The peak of the response of the newly allocated unit is set to the novel input,

$$\vec{c}_n = \vec{I}. \tag{2.4}$$

The linear synapses on the second layer are set to the difference between the output of the network and the novel output,

$$\vec{h}_n = \vec{T} - \vec{y}. \tag{2.5}$$

The width of the response of the new unit is proportional to the distance from the nearest stored vector to the novel input vector,

$$w_n = \kappa ||\vec{I} - \vec{c}_{\text{nearest}}||. \tag{2.6}$$

where κ is an overlap factor. As κ grows larger, the responses of the units overlap more and more.

The RAN uses a two-part novelty condition. An input-output pair $(\vec{I}, \vec{T})$ is considered novel if the input is far away from existing centers,

$$||\vec{I} - \vec{c}_{\text{nearest}}|| > \delta(t), \tag{2.7}$$

and if the difference between the desired output and the output of the network is large

$$||\vec{T} - \vec{y}(\vec{I})|| > \epsilon. \tag{2.8}$$

Typically, ϵ is a desired accuracy of output of the network. Errors larger than ϵ are immediately corrected by the allocation of a new unit, while errors smaller than ϵ are gradually repaired using gradient descent. The distance $\delta(t)$ is the scale of resolution that the network is fitting at the tth input presentation. The

learning starts with $\delta(t) = \delta_{\max}$, which is the largest length scale of interest, typically the size of the entire input space of non-zero probability density. The distance $\delta(t)$ shrinks until the it reaches $\delta_{\min}$, which is the smallest length scale of interest. The network will average over features that are smaller than $\delta_{\min}$. We used a function:

$$\delta(t) = \max(\delta_{\max} \exp(-t/\tau), \delta_{\min}), \tag{2.9}$$

where τ is a decay constant.

At first, the system creates a coarse representation of the function, then refines the representation by allocating units with smaller and smaller widths. Finally, when the system has learned the entire function to the desired accuracy and length scale, it stops allocating new units altogether.

The two-part novelty condition is necessary for creating a compact network. If only condition (2.7) is used, then the network will allocate units instead of using gradient descent to correct small errors. If only condition (2.8) is used, then fine-scale units may be allocated in order to represent coarse-scale features, which is wasteful.

By allocating new units the RAN eventually represents the desired function ever more closely as the network is trained. Fewer units are needed for a given accuracy if the first-layer synapses c_{jk}, the second-level synapses h_{ij}, and the thresholds γ_i are adjusted to decrease the error:

$$\mathcal{E} = ||\vec{y} - \vec{T}||^2. \tag{2.10}$$

We use the Widrow-Hoff LMS algorithm (Widrow & Hoff, 1960) to decrease the error whenever a new unit is not allocated:

$$\begin{aligned} \Delta\vec{h}_j &= \alpha(\vec{T} - \vec{y})x_j, \\ \Delta\vec{\gamma} &= \alpha(\vec{T} - \vec{y}) \end{aligned} \tag{2.11}$$

In addition, we adjust the centers of the responses of units to decrease the error:

$$\Delta c_{jk} = 2\frac{\alpha}{w_j}(I_k - c_{jk})x_j \left[(\vec{T} - \vec{y}) \cdot \vec{h}_j\right]. \tag{2.12}$$

Equation (2.12) is derived from gradient descent and equation (2.1). Equation (2.12) also has an intuitive interpretation. Units whose outputs that would cancel the error have their centers pulled towards the input. Units whose outputs that would increase the error have their centers pushed away from the input. Empirically, equation (2.12) also works for the polynomial approximation (2.2).

The structure of the algorithm is shown below as pseudo-code, including initialization code:

$\delta = \delta_{\max}$
$\vec{\gamma} = \vec{T}_0$ (from the first input-output pair)
loop over presentations of input-output pairs $(\vec{I}, \vec{T})$
    {
        evaluate output of network $\vec{y} = \sum_j \vec{h}_j x_j(\vec{I}) + \vec{\gamma}$
        compute error $\vec{E} = \vec{T} - \vec{y}$
        find distance to nearest center $d = \min_j ||\vec{c}_j - \vec{I}||$
        if $||\vec{E}|| > \epsilon$ and $d > \delta$ then
            {
                allocate new unit, $\vec{c}_{\text{new}} = \vec{I}, \vec{h}_{\text{new}} = \vec{E}$
                if this is the first unit to be allocated then
                    width of new unit $= \kappa\delta$
                else
                    width of new unit $= \kappa d$
            }
        else
            perform gradient descent on $\vec{\gamma}, \vec{h}_j, c_{jk}$
        if $\delta > \delta_{\min}$
            $\delta = \delta * \exp(-1/\tau)$
    }

3 Results

One application of an interpolating RAN is to predict complex time series. As a test case, a chaotic time series can be generated with a nonlinear algebraic or differential equation. Such a series has some short-range time coherence, but long-term prediction is very difficult. The need to predict such a time series arises in such real-world problems as detecting arrhythmias in heartbeats.

The RAN was tested on a particular chaotic time series created by the Mackey-Glass delay-difference equation:

$$x(t+1) = (1-b)x(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}}, \tag{3.1}$$

for $a = 0.2$, $b = 0.1$, and $\tau = 17$.

The network is given no information about the generator of the time series, and is asked to predict the future of the time series from a few samples of the history of the time series. In our example, we trained the network to predict the value at time $T + \Delta T$, from inputs at time T, $T - 6$, $T - 12$, and $T - 18$.

The network was tested using two different learning modes: off-line learning with a limited amount of data, and on-line learning with a large amount of data. The Mackey-Glass equation has been learned off-line, by other workers, using the back-propagation algorithm (Lapedes & Farber), and radial basis functions (Moody & Darken, 1989). We used RAN to predict the Mackey-Glass equations with the following parameters: $\alpha = 0.02$, 400 learning epochs, $\delta_{\max} = 0.7$, $\kappa = 0.87$ and $\delta_{\min} = 0.07$ reached after 100 epochs. RAN was simulated using $\epsilon = 0.02$ and $\epsilon = 0.05$. In all cases, $\Delta T = 85$.

Figures 2 and 3 compares the RAN to the other learning algorithms. Figure 2 shows the normalized error rate on a test set versus the size of the learning set for various algorithms. The test set is 500 points of the output of the Mackey-Glass equation at $T = 4000$. The normalized error is the RMS error divided by the sqaure root of the variance of the output of the Mackey-Glass equation.

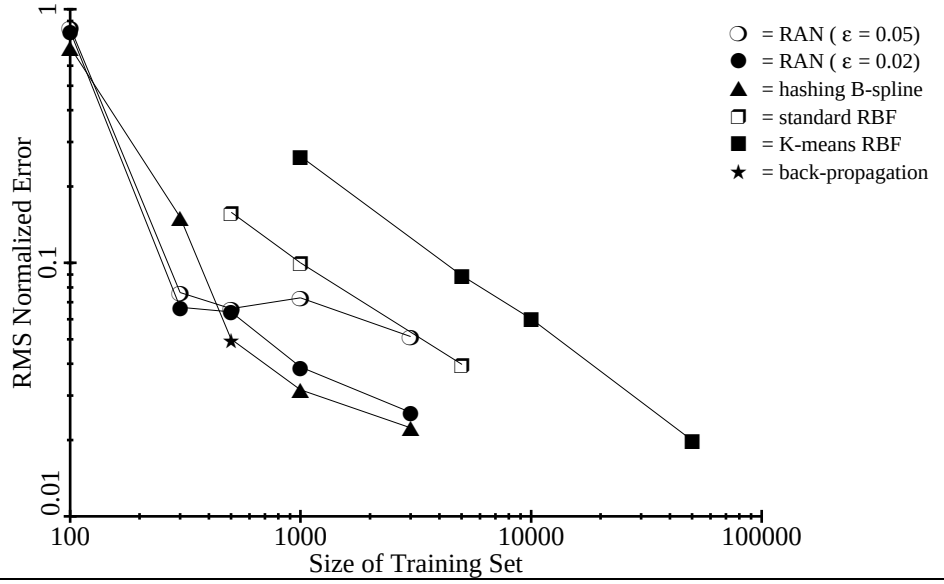

Figure 2: The normalized RMS error on a test set for various off-line learning algorithms. Back-propagation, RAN, and hashing B-splines are all competitive in error rate. (Near the back-propagation symbol, the symbol for hashing B-splines is omitted for clarity).

When the RAN algorithm is optimized for accuracy ($\epsilon = 0.02$), then it attains accuracy comparable to hashing B-splines. Figure 3 shows the size of the network versus the size of the learning set. As the size of the learning set grows, the number of units allocated by RAN grows very slowly. The size of the network is measured via number of weights or parameters, which is a approximation to the complexity of

the network. For back-propagation, the size is the number of synapses. For the RBF networks and for RAN, there are six parameters per unit: four to describe the location of the center, one for the width, and one for the height of the Gaussian. For hashing B-splines, each unit has two parameters: the hash table index and its corresponding hash table value.

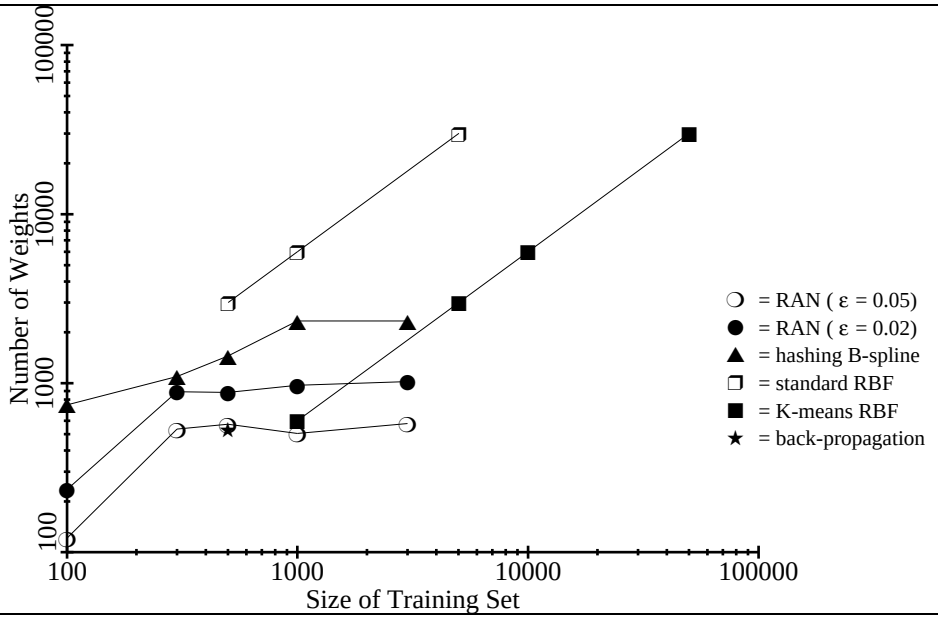

Figure 3: The number of weights in the network versus the size of the training set. RAN and back-propagation are competitive in the compactness of the network. Notice that as the training set size increases, the size of the RAN stays roughly constant.

Figure 4 shows the efficiency of the various learning algorithms: the smallest, most accurate algorithms are towards the lower left. When optimized for size of network ($\epsilon = 0.05$), the RAN has about as many weights as back-propagation and is just as accurate. The efficiency of RAN is roughly the same as back-propagation, but requires much less computation: RAN takes approximately 8 minutes of SUN-4 CPU time to reach the accuracy listed in figure 4, while back-propagation took approximately 30–60 minutes of Cray X-MP time.

The novelty criteria and the center adjustment are both important to the performance of the RAN algorithm. We tested off-line learning of Mackey-Glass predictions using three styles of network that share the same transfer function: a flat network whose centers are chosen with the K-means algorithm, a hierarchical network whose centers are chosen with the K-means algorithm, and a RAN. Each of these networks were tested with either center adjustment via gradient descent or no center adjustment at all. Table 1 shows the normalized RMS error on a test set after training off-line on 500 examples. The non-hierarchical K-means network was formed with 100 units. The hierarchical K-means network was formed with three sets of centers: K-means was run separately for 75, 20, and 5 units. In both K-means networks, the widths of the units were chosen via equation (2.6), with a $\kappa = 0.87$. Using the same parameters as used above, and with $\epsilon = 0.05$, the RAN allocated 100 units without center adjustment, and 95 units with center adjustment.

	Flat Network	Hierarchical Network	RAN
no center adjust	0.54	0.31	0.17
center adjust	0.20	0.15	0.066

Table 1: Normalized RMS Error for various sub-strategies of RAN

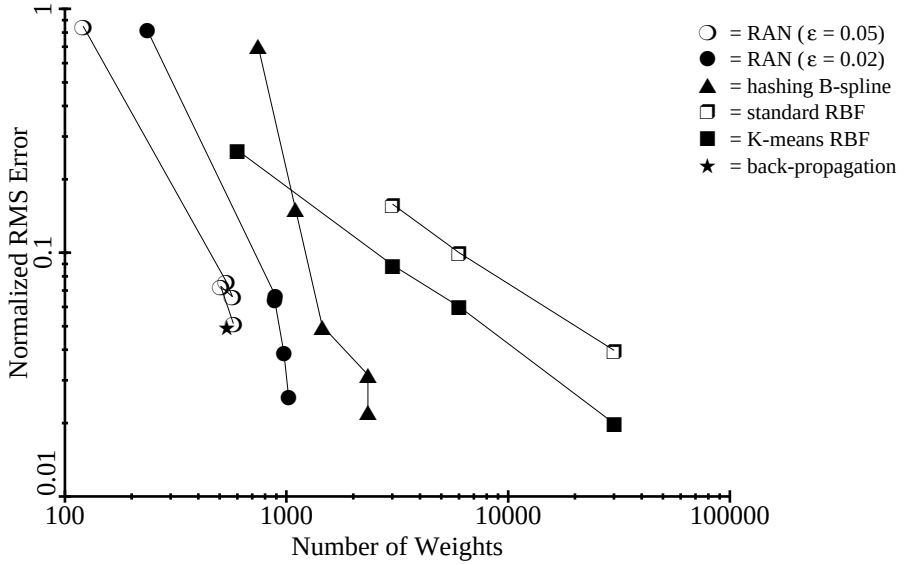

Figure 4: The error on a test set versus the size of the network. Back-propagation stores the prediction function very compactly and accurately, but takes a large amount of computation to form the compact representation. RAN is as compact and accurate as back-propagation, but uses much less computation to form its representation.

Table 1 shows that the three sub-strategies of RAN are about equally important. Using hierarchy, adjusting the centers via gradient descent, and choosing units to allocate based on the novelty conditions all seem to improve the performance by roughly a factor of 1.5 to 2.

The Mackey-Glass equation has been learned using on-line techniques by hashing B-splines (Moody, 1989). We used on-line RAN using the following parameters: $\alpha = 0.05$, $\epsilon = 0.02$, $\delta_{\max} = 0.7$, $\delta_{\min} = 0.07$, $\kappa = 0.87$, and $\delta_{\min}$ reached after 5000 input presentations. Table 2 compares the on-line error versus the size of network for both RAN and the hashing B-spline (Moody, personal communication). In both cases, $\Delta T = 50$. The RAN algorithm has similar accuracy to the hashing B-splines, but the number of units allocated is between a factor of 2 and 8 smaller.

Method	Number of Units	Normalized RMS Error
RAN	143	0.054
Hashing B-spline 1 level of hierarchy	284	0.074
Hashing B-spline 2 levels of hierarchy	1166	0.044

Table 2: Comparison between RAN and hashing B-splines

Table 3 shows the effectiveness of the ϵ novelty condition for on-line learning. When ϵ is set very low, the network performs very well, but is very large. Raising ϵ decreases the size of the network without substantially affecting the performance of the network. For $\epsilon > 0.05$, the network becomes very compact, but the accuracy becomes poor.

ϵ	Number of Units	Normalized RMS Error
0	189	0.055
0.01	174	0.050
0.02	143	0.054
0.05	50	0.071
0.10	26	0.102

Table 3: Effectiveness of ϵ novelty condition

Figure 5 shows the output of the RAN after having learned the Mackey-Glass equation on-line. In the simulations, the network learns to roughly predict the time series quite rapidly. Notice in figure 5a the sudden jumps in the output of the network, which show that a new unit has been allocated. As more examples are shown, the network allocates more units and refines its predictions.

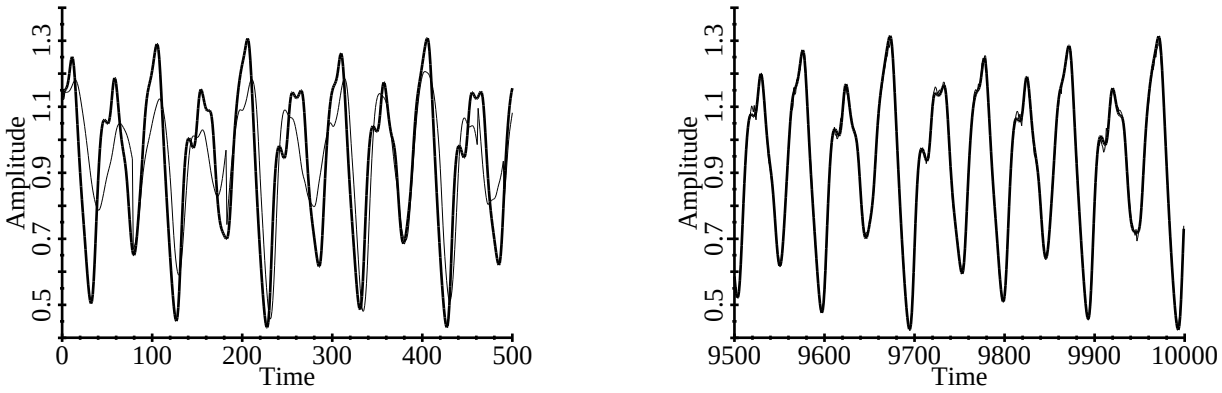

Figure 5: The output of the RAN as it learns on-line. The thick line is the output from the Mackey-Glass equation, the thin line is the prediction by the network. On the left, figure 5a shows the beginning of the learning. Very quickly, RAN picks up the basic oscillatory behavior of the Mackey-Glass equation. On the right, figure 5b shows the end of the on-line learning. At $T = 10000$, the predictions match the actual output very well.

4 Conclusions

There are various desirable attributes for a network that learns: it should learn quickly, it should learn accurately, and it should form a compact representation. Formation of a compact representation is particularly important for networks that are implemented in hardware, because silicon area is at a premium. A compact representation is also important for statistical reasons: a network that has too many parameters can overfit data and generalize poorly.

Many previous network algorithms either learned quickly at the expense of a compact representation, or formed a compact representation only after laborious computation. The RAN is a network that can find a compact representation with a reasonable amount of computation.

Acknowledgements

Thanks to Carver Mead, Carl Ruoff, and Fernando Pineda for useful comments on the paper. Thanks to Glenn Gribble for helping to put the paper together. Special thanks to John Moody who not only provided useful comments on the paper, but also provided data on the hashing B-splines.

References

Baum, E. B., 1989, A Proposal for More Powerful Learning Algorithms, *Neural Computation,* **1(2),** 201–207.

Broomhead, D., Lowe, D., 1988, Multivariable function interpolation and adaptive networks, *Complex Systems,* **2,** 321–355.

Judd, S., 1988, On the Complexity of Loading Shallow Neural Networks, *Journal of Complexity,* **4.**

Lapedes, A., Farber, R., 1987, *Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling,* Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM.

Lloyd, S. P., 1957, Least squares quantization in PCM. Bell Laboratories Internal Technical Report.

MacQueen, J., 1967, Some Methods for Classification and Analysis of Multivariate Observations. *In:* Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probability, eds. L. M. LeCam and J. Neyman. Berkeley: U. California Press, 281.

Moody, J, Darken, C., 1988, Learning with Localized Receptive Fields, *In:* Proceedings of the 1988 Connectionist Models Summer School, D. Touretzky, G. Hinton, T. Sejnowski, eds., 133–143, Morgan-Kaufmann, San Mateo.

Moody, J, Darken, C., 1989, Fast Learning in Networks of Locally-Tuned Processing Units, *Neural Computation,* **1(2),** 281–294.

Moody, J., 1989, Fast Learning in Multi-Resolution Hierarchies, *In:* Advances in Neural Information Processing Systems I, D. Touretzky, ed., 29–39, Morgan-Kaufmann, San Mateo.

Poggio, T., Girosi, F., 1990, Regularization Algorithms for Learning that are Equivalent to Multilayer Networks, *Science,* **247,** 978–982.

Powell, M. J. D., 1987, Radial Basis Functions for Multivariable Interpolation: A Review, *In:* Algorithms for Approximation, J. C. Mason, M. G. Cox, eds., Clarendon Press, Oxford.

Stark, L., Okajima, M., Whipple, G. H., 1962, Computer Pattern Recognition Techniques: Electrocardiographics Diagnosis, *Comm. of the ACM,* **5**, 527–532.

Widrow, B., Hoff, M., 1960, Adaptive Switching Circuits, *In:* 1960 IRE WESCON Convention Record, 96–104, IRE, New York.