

Note that all of the feedback in the augmented architecture takes place in a single layer of neurons. If this restriction is removed so that feedback may be provided after several feedforward layers, then clearly any finite-state recognizer can be implemented.

We show that while the augmented architecture can be used to recognize strings with parity, at least three different state vectors are needed to handle this problem, for which the minimal machine has only two states, as shown in Fig. 2(a). This demonstrates that, while minimal representations may not be possible, non-minimal representations may exist.

1) *Theorem 3*: An augmented first-order SLRNN can not recognize all strings that have parity if the network is only allowed to utilize two state vectors; i.e., if the network must implement the minimal finite-state recognizer.

Proof: The proof of Theorem 3 is essentially identical to that of Theorem 1, but in this case there are two different state vectors as opposed to two different output vectors.

But we can show by example that the augmented architecture can solve the parity problem if three different state vectors are allowed. The example actually implements the non-minimal finite-state recognizer shown in Fig. 2(b). States *C* and *D* of Fig. 2(b) are equivalent to state *A* in Fig. 2(a), while state *E* in Fig. 2(b) is equivalent to state *B* in Fig. 2(a). This phenomenon is called *state-splitting*, since state *A* "splits" into states *C* and *D*.

Let $M = 2$ and $N = 2$. Now let $\mathbf{S}_0 = [0, 1]^T$ (for state *C*), $\mathbf{S}_1 = [1, 0]^T$ (for state *D*), and $\mathbf{S}_2 = [1, 1]^T$ (for state *E*). \mathbf{S}_0 is the initial state vector. The input vectors are $\mathbf{I}_0 = [1, 0]^T$ and $\mathbf{I}_1 = [1, 1]^T$. Under these conditions, it can easily be verified that the augmented first-order SLRNN with the following weights recognizes odd parity strings:

$$\begin{aligned} w_{11} &= 1, & w_{12} &= 1, & w_{13} &= -\frac{3}{2}, & w_{14} &= 1, \\ w_{21} &= -1, & w_{22} &= -1, & w_{23} &= \frac{5}{2}, & w_{24} &= -1. \end{aligned} \quad (11)$$

This implementation of odd parity requires only two state neurons, while Minsky's implementation would require four [4].

IV. CONCLUSION

We have shown that a second-order SLRNN can implement any finite-state recognizer, while a first-order SLRNN can not. This is an example of the improved representational ability achieved by utilizing a second-order network. On the other hand, a second-order SLRNN has N^2M weights, while a first-order SLRNN has only $N(M+N)$.

We have also shown that an augmented first-order SLRNN architecture can handle the parity problem. In this case, we allowed feedforward output layers of neurons. Intriguingly, the augmented first-order SLRNN was proven to be unable to implement the minimal parity recognizer, but it could implement a non-minimal parity recognizer. It thus becomes obvious that state-splitting is an important method that can be used to expand the representational abilities of augmented first-order SLRNN's.

REFERENCES

[1] S. Fahlman, "The recurrent cascade-correlation architecture," in *Advances in Neural Information Processing Systems 3*, R. Lippmann, J. Moody, and D. Touretzky, Eds., San Mateo, CA: Morgan Kaufmann Publishers, 1991, pp. 190-196.
 [2] C. Giles, C. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, pp. 393-405, 1992.

[3] N. Alon, A. Dewdney, and T. Ott, "Efficient simulation of finite automata by neural nets," *J. Ass. Computing Machinery*, vol. 38, no. 2, pp. 495-514, 1991.
 [4] M. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1967, ch. 3, pp. 32-66.
 [5] D. Seidl and R. Lorenz, "A structure by which a recurrent neural network can approximate a nonlinear dynamic system," in *Proc. Int. Joint Conf. Neural Networks 1991*, vol. II, pp. 709-714, July 1991.
 [6] H. Siegelmann and E. Sontag, "On the computational power of neural nets," in *Proc. Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, PA, July 1992.
 [7] L. Atlas and Y. Suzuki, "Digital systems for artificial neural networks," *IEEE Circuits Devices Mag.*, vol. 5, pp. 20-24, 1989.
 [8] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179-211, 1990.
 [9] J. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, no. 2/3, pp. 227-252, 1991.
 [10] R. Watrous and G. Kuhn, "Induction of finite-state languages using second-order recurrent networks," in *Advances in Neural Information Processing Systems 4*, J. Moody, S. Hanson, and R. Lippmann, Eds., San Mateo, CA: Morgan Kaufmann Publishers, 1992, pp. 309-316.
 [11] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978, second ed.

SVD-NET: An Algorithm that Automatically Selects Network Structure

Dimitris C. Psychogios and Lyle H. Ungar

Abstract—An algorithm is developed for training feedforward neural networks that uses singular value decomposition (SVD) to identify and eliminate redundant hidden nodes. Minimizing redundancy gives smaller networks, producing models that generalize better and thus eliminate the need of using cross-validation to avoid overfitting. The method is demonstrated by modeling a chemical reactor.

I. INTRODUCTION

Neural networks typically contain a large number of adjustable parameters (called *weights*) that, given an appropriate model structure, enables them to approximate training data with arbitrary accuracy. However, good fit on training data does not assure good performance on future data, especially if the number of adjustable parameters is large compared to the available training data. One estimate of future performance is the predicted squared error [1], which is determined as the sum of squared error on the training set and a model complexity penalty based on the number of adjustable parameters. Training error decreases as the number of model parameters increases, but the model complexity increases. Therefore, even though models with a large number of parameters may achieve quite low error on the training data, prediction error on new data may be high as a result of the penalty term for model complexity. In practice, simpler models are expected to perform better (i.e., generalize better) on future data from the same distribution than more complex ones.

In neural networks, the problem of high prediction error on future data is commonly referred to as overfitting. One method to avoid overfitting is a simplified version of cross-validation: The available

Manuscript received October 4, 1992; revised December 8, 1992.
 The authors are with the Department of Chemical Engineering, University of Pennsylvania, Philadelphia, PA 19104-6393, USA.
 IEEE Log Number 9207232.

data are split in two subsets (training and testing) and the network model is trained until the minimum error on the testing set is achieved. However, this procedure does not assure optimal performance on future data, as it is applied to a fixed-size network that may be redundant. Many *ad hoc* techniques have been proposed to prune redundant nodes from networks, but network model selection is largely done by testing a range of network sizes, i.e., number of hidden nodes. We propose a new algorithm (SVD-NET) for network training and model selection that, starting from an arbitrary feedforward network model structure, automatically determines the optimal number of hidden nodes, given the available data. Conjugate gradient optimization is used to determine the internal weights of the network, and singular value decomposition (SVD) is used to calculate the weights from the final hidden layer to the network's outputs, and at the same time identify redundant hidden nodes. These nodes are eliminated and the reduced-size network is retrained until convergence. The SVD-NET method has the virtue of being both simple and rigorous.

II. TRAINING ALGORITHM

Consider a single hidden layer, fully connected network whose equational form is

$$y_k = \sum_i^M b_{ik} \cdot \sigma_i(\mathbf{a}_i^T \cdot \mathbf{x} + w_i) \quad (1)$$

where y_k is the k th output, M is the number of hidden nodes, b_{ik} is the weight from hidden node i to output node k , \mathbf{a}_i^T is a column vector of input weights to the i th hidden node (augmented by a constant bias term w_i), \mathbf{x} is the input vector to the network and σ is the nonlinear transformation performed by a hidden node, such as the sigmoid:

$$\sigma_i = \frac{1 - \exp(-\mathbf{a}_i^T \cdot \mathbf{x} + w_i)}{1 + \exp(-\mathbf{a}_i^T \cdot \mathbf{x} + w_i)} \quad (2)$$

The SVD-NET algorithm alternates between estimation of the weights from input to hidden nodes (\mathbf{a}_i^T) and the weights from hidden to output nodes (b_{ik}) in a two-stage process as follows: 1) Given a set of values for the b_{ik} coefficients, the weights \mathbf{a}_i^T are adjusted using a nonlinear optimization method to minimize the sum of squared errors. During this stage the coefficients b_{ik} remain fixed. 2) Given the new values for the weights from input to hidden nodes, the outputs of the hidden nodes (the functions σ_i in (2)) are calculated. These outputs, which are nonlinear transformations of the inputs, are used to calculate a new set of coefficients b_{ik} using linear least squares on (1).

Use of least-squares techniques, more specifically SVD, allows well-developed linear methods to be applied to the training and selection of appropriate sizes of neural networks. Linear projection techniques, such as partial least squares (PLS), have been used to improve network models, but the number of hidden nodes was still determined in an *ad hoc* manner [2] or through simplified cross-validation [3]. We use SVD as an explicit model selection method (i.e., to identify and remove redundant hidden nodes during network training) and not *a priori*. The advantage of such an approach is that it adapts the model structure (number of hidden nodes) to the available data and avoids the need for potentially expensive cross-validation procedures, thus allowing all available data to be used for training.

A. Nonlinear Optimization

The conjugate gradient minimization algorithm combined with a line search minimization is used to adjust the input weights (\mathbf{a}_i^T). For a detailed discussion on the intuition and theoretical derivation of the

method see [4], [5]; an example of the application of the method to network training is given in [6]. In our formulation, the objective function is the sum of squared errors

$$E = \frac{1}{2} \cdot \sum_j^N \sum_k^P (y'_{jk} - y_{jk})^2 \quad (3)$$

where the sum is over all N training examples and P outputs; y' represents the actual response values and y represents the model's outputs (predictions). The neural network model is described by (1), with activation functions given by (2). The Polak-Ribiere update formula is used to calculate conjugate directions, given the gradient of (3) with respect to a weight a_{mi} from input node m to hidden node i . Iterations continue until the change in the objective function between iterations is below a prespecified threshold or the gradient is exactly zero, which implies that the optimum has been reached.

B. Singular Value Decomposition

Once the input weights a_{mi} are known, the output of each hidden node for each input pattern j can be calculated. This gives an $N \times M$ matrix, where N is the number of observations and M is the number of hidden nodes. Then we can formulate the following linear least squares problem: Calculate b_k such that the residual

$$E_k = \|A \cdot b_k - y'_k\| \quad (4)$$

is minimized, where A is the $N \times M$ matrix having as columns the outputs of each hidden node for all N input vectors and y'_k is a vector of the k th response values. This linear least squares formulation can be obtained for all P responses.

The basic premise behind SVD is that a general $N \times M$ matrix can be decomposed in three matrices U , W , and V as

$$A = U \cdot W \cdot V^T \quad (5)$$

where U is an $N \times M$ column-orthogonal matrix, W is an $N \times M$ diagonal matrix with positive or zero elements and V is an $M \times M$ orthogonal matrix. The elements of the W matrix are called the singular values of the A matrix. For a detailed discussion on the implementation and interpretation of SVD see [4] and [7]. The matrix A may be singular (or nearly singular) because of row or column degeneracies. Column degeneracies imply that the hidden node outputs are correlated. This means that the problem is overdetermined with the given set of hidden nodes; i.e., the available data cannot help distinguish between them. In practical terms this suggests the presence of redundant hidden nodes.

This observation motivates the use of singular value decomposition as the least squares solution technique in our network training method. The development of a model network involves choice of an *a priori* fixed number (typically large) of hidden nodes, with the hope that they will be adequate to approximate the unknown underlying function. However, it may well be the case that this initial model is over-parameterized (redundant). Redundant hidden nodes cause singularities in the A matrix, which can be identified through inspection of its singular values. A nonzero number of small singular values indicates redundancy in the initial choice for the number of hidden layer nodes. The algorithm then eliminates these hidden nodes and retains the "pruned" network model.

The SVD-NET algorithm presented above is for networks with one hidden layer, but can be extended to develop multilayered networks. This is simply done by considering the outputs of the first hidden layer as new variables to be used as inputs to the second hidden layer. Then, the SVD-NET algorithm can be applied exactly as described in Section II-A. In this way, the algorithm essentially develops a multilayered network where the number of hidden nodes in each hidden layer is automatically determined.

TABLE I
RMSE AND STANDARD DEVIATION FOR NEURAL NETWORK MODELS
TRAINED WITH THE SVD-NET AND BACK-PROPAGATION ALGORITHMS

	SVD-NET Training	Back-Propagation Training
CSTR, full state	0.0077 ± 0.0021	0.0201 ± 0.0044
CSTR, ARMA	0.0124 ± 0.0028	0.0251 ± 0.0050

III. APPLICATION EXAMPLES

The SVD-NET algorithm was tested on two variations of a test problem. Results, in terms of prediction error, were compared with networks trained using the standard backpropagation algorithm. In the first test problem, the output concentration of a nonisothermal continuously stirred tank reactor (CSTR) is predicted based on measurements of the state and control variables [8]. In the second variation, an ARMA-type model of the dynamic system was developed, where the predicted output is a function of the current and two past values of the output and the system's control input (the inlet feed temperature T_i). In each case, fifty training sets, each containing 250 data points, were generated. Two additional data sets with true responses (where the outputs were not corrupted with noise), were generated in order to test the network's predictions.

The initial networks for the first test problem had four inputs, one hidden layer of thirty nodes, and one output node; for the second example, the number of input nodes was increased to six. Networks trained with the SVD-NET algorithm used all 250 data points, whereas for networks trained with backpropagation the simplified cross-validation technique (see introduction) was used to avoid overfitting. The root-mean-squared error (RMSE) was calculated for each network model by using the noise-free data set for all 50 network models and for both test problems. A mean and standard deviation of the RMSE were also calculated, and are reported below for both SVD-NET and backpropagation.

The network models trained with the SVD-NET method have a much smaller RMSE than networks trained using backpropagation, and smaller standard deviation. This is because the SVD-NET method eliminates a significant number of hidden nodes during training; thus, the final structure of the network model is adapted to the data at hand and has significantly fewer adjustable parameters. For both test problems, the final network model had between 8 and 10 hidden nodes for all 50 data sets, with the vast majority of models having 9 hidden nodes. As explained in the introduction, when the number of adjustable parameters is decreased, prediction accuracy in future data is expected to improve. Cross-validation to determine when to stop the weight updates partially prevents overfitting, in the sense that the network is trained to partial convergence, but it does not select the appropriate model structure, and hence the model may still be over-parameterized (redundant). The SVD-NET algorithm eliminates this redundancy. This means that networks trained with the SVD-NET algorithm are expected to generalize better than networks with fixed structure. Though a complete training session involves training of several networks (of decreasing size), the SVD-NET algorithm also offers a speed-up of up to an order of magnitude compared to standard neural networks.

IV. DISCUSSION

This paper presents a new methodology for training neural networks: the SVD-NET algorithm. The two key features in this method are the development of the network model by training (adjusting weights) one layer at a time, and the automatic elimination of

redundant hidden nodes using SVD. This is accomplished through examination of the singular values of the matrix of hidden node outputs, since correlated hidden node outputs result in the matrix having small singular values. As many hidden nodes are eliminated as there are small singular values, and the new "pruned" network is retrained. In this way, we can obtain a final model with fewer adjustable parameters and more accurate predictions than a network model with a fixed, *a priori* determined, size. Thus, the SVD-NET algorithm is an alternative to network pruning techniques [9], [10], which are mostly heuristic and remove nodes and weights via arbitrary relevance measures or by examination of their impact on the network's predictions.

The SVD-NET algorithm explicitly addresses the problem of overfitting by selecting model structure using all the available data; as the number of hidden nodes is optimized, the network can be trained to full convergence. In this context it compares favorably to simplified cross-validation that uses part of the data to determine model structure ([2], [3]) or to decide when to stop training [8]. As was demonstrated in this paper, the latter approach does not prevent overfitting, and only uses part of the data for training.

REFERENCES

- [1] A. R. Barron and R. L. Barron, "Statistical learning networks: A unifying view," *Computing Science and Statistics. Proc. Twentieth Symp. on the Interface*, E. J. Wegman, D. T. Gantz, and J. J. Miller, Eds. Alexandria, VA: Amer. Stat. Association, pp. 192-203.
- [2] T. R. Holcomb and M. Morari, "PLS/neural networks," *Comp. Chem. Eng.*, vol. 16, pp. 393-411, 1992.
- [3] S. J. Qin and T. J. McAvoy, "Nonlinear PLS modeling using neural networks," *Comp. Chem. Eng.*, vol. 16, pp. 379-391, 1992.
- [4] W. H. Press, B. P. Flannery, S. A. Teulosky, and W. T. Vetterling, *Numerical Recipes*. Cambridge, UK: Cambridge University Press, 1988.
- [5] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. San Diego, CA: Academic Press, 1981.
- [6] J. Leonard and M. A. Kramer, "Improvement of the backpropagation algorithm for training neural networks," *Comp. Chem. Eng.*, vol. 14, pp. 337-341, 1990.
- [7] N. L. Ricker, "The use of biased least-squares estimates for parameters in discrete-time pulse response models," *Ind. Eng. Chem. Res.*, vol. 27, pp. 343-350, 1988.
- [8] D. C. Psychogios and L. H. Ungar, "Direct and indirect model-based control using artificial neural networks," *Ind. Eng. Chem. Res.*, vol. 30, pp. 2564-2573, 1991.
- [9] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989.
- [10] E. D. Karnin, "A simple procedure for pruning backpropagation trained neural networks," *IEEE Trans. on Neural Networks*, vol. 1, pp. 239-242, 1990.