Interpolation Results in the Split Inversion Neural Network Algorithm

Alastair D. McAulay and Ramesh Ravula

Wright State University, Department of Computer Science and Engineering
Dayton, Ohio 45435

ABSTRACT

An advantage of a neural network over a look-up table is the ability to act as a higher form of associative memory. For example, a network in addition to giving correct results for the pattern examples that are used to train the network, can also perform sophisticated interpolation. In this paper neural networks trained with the split inversion neural network algorithm are shown to provide complex interpolation when used as associative memories. The neural network is applied to the design of a beam-truss-spring structure. The results obtained using the neural network algorithm are shown to be superior to those of a look-up table.

INTRODUCTION

Computer aided engineering design tools are good for analyzing a hypothesised design but are of little help in creating the design in the first place. Expert systems could be applied to the design task but these are proving to be cumbersome. This is because whenever an expert increases his expertise, new rules must be added to the existing system that do not conflict with them. Addition of more rules makes the system slower and less efficient.

Neural networks have advantages for engineering design. The concept of associative memory which is an integral part of human expertise is inherent in neural networks [1]. Associative memories unlike conventional memories can associate a new situation with previous ones. The neural network provides a higher form of associative memory than a look-up table because of its ability to provide complex interpolation.

It will be shown in this paper that the split inversion neural network algorithm can exhibit this higher form of associative memory when applied to the design of engineering systems. An example from structural design, a beam-truss-spring, is used to illustrate this property. The results obtained with the neural network algorithm and the look up table are discussed, compared and the superiority of neural networks illustrated.

SPLIT INVERSION LEARNING ALGORITHM

The concept for the split inversion algorithm was explained in reference [2]. The considerable speed advantage of this algorithm over the backpropagation algorithm was described by the author in reference [3]. The split inversion algorithm in contrast to the backpropagation algorithm [4] computes the weights for both the output and the hidden layers so as to minimize the square error at the output of the network [5].

A least square solution for a weight update-increment vector $\mathbf{\Delta w}$ may be computed for all weights at the same time using conjugate gradients [5] from

$$\mathbf{A\Delta w} = \mathbf{d} \tag{1}$$

where $\mathbf{A}$ is computed from derivatives of the outputs with respect to the weights and $\mathbf{d}$ is an output error vector. The weights are updated at the l th iteration

$$w_{ij}(l+1) = w_{ij}(l) + \Delta w_{ij}(l) \tag{2}$$

APPLICATION TO BEAM-TRUSS-SPRING DESIGN

A beam-truss-spring is considered from structural engineering, figure 1. It is simple to compute the deflections $\mathbf{u}$ under a loading $\mathbf{p}$ for a specific structure for which the lengths of the members $\mathbf{l}$ and their stiffnesses $\mathbf{s}$ are known.

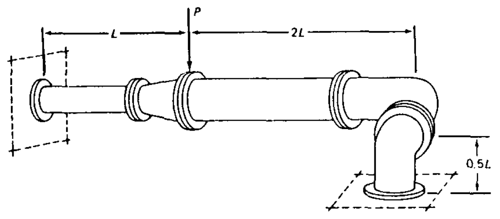

Figure 1: Structural beam-truss-spring design example

$$\mathbf{B}=\frac{1}{\ell^3}\begin{bmatrix} 12s_1 & -6s_1\ell & -12s_1 & -6s_1\ell & 0 & 0 & 0 \\ & 4s_1\ell^2 & 6s_1\ell & 2s_1\ell^2 & 0 & 0 & 0 \\ & & 12(s_1+s_2) & (6s_1-12s_2)\ell & -12s_2 & -12s_2\ell & 0 \\ & & & (4s_1+16s_2)\ell^2 & 12s_2\ell & 8s_2\ell^2 & 0 \\ & & & & 2s_4\ell^2+12s_2 & 12s_2\ell & -2s_4\ell^2 \\ & \text{Symmetric} & & & & s_3+16s_2\ell^2 & 0 \\ & & & & & & 2s_4\ell^2 \end{bmatrix}$$

Figure 2: Stiffness matrix for design example

This is an analysis problem involving solving the linear equations for deflections **u** given a design **B** and a loading **p**

$$\mathbf{B}(\mathbf{l},\mathbf{s})\mathbf{u}=\mathbf{p} \qquad (3)$$

The inverse or design problem of determining the lengths of the members **l** and/or their stiffnesses **s** for a given load and deflection is a very much harder problem. In general the solution is not unique and may not be computable. The reason is that the design parameters are distributed throughout the equations and may arise in a complicated manner, e.g. length arises with powers in **B**, figure 2. The uniqueness issue is bypassed by fixing certain parameters.

A neural network may be constructed and trained to perform the inversion or design problem. The sequence of actions taken to train the neural network are as follows:

1. Compute the matrix **B** in equation (3), figure 2, using a selected stiffness vector **s** and length l.
2. Use equation (3) with load vector **p** to compute corresponding deflection vector **u**.
3. Repeat 1 and 2 with other design parameters **s** and l and with other load vectors **p**.
4. Associated vectors are determined where load and deflection vectors are concatenated to form an input vector and the corresponding stiffness vector and length form the output vector.
5. The network is trained using the split inversion algorithm for these sets of input and output vectors.

The network will now provide at the output a length and stiffness vector given a deflection vector and load vector at the input. Further, the deflection vector and load vector need not be one of those used for training. The network will interpolate to handle those that fall between training cases.

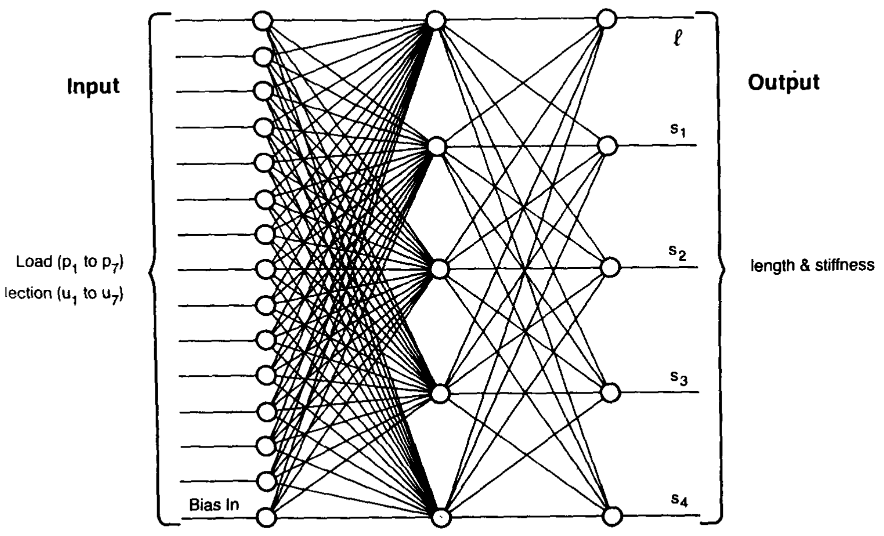

Figure 3: Neural network for structural example

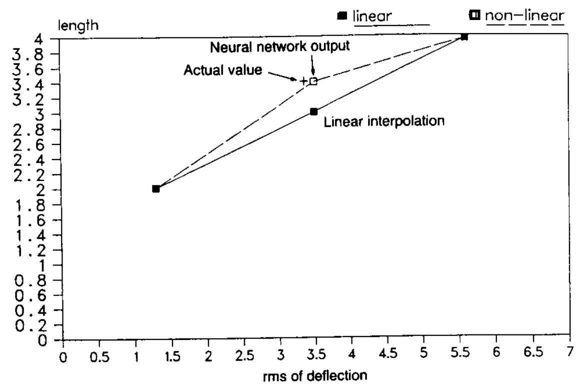

Figure 4: Illustrates interpolation for neural network

SAMPLE RESULT SHOWING INTERPOLATION

The neural network used to accomodate the correct number of inputs and outputs for the beam-truss-spring in figure 1 is shown in figure 3. The network is trained with four design examples. In the recall phase, if the network is given any input pattern that it has been trained with, the network responds with the corresponding output pattern. The question addressed next is the behavior when an input is provided that was not one used in training.

In order to demonstrate interpolation, the stiffness of the structure and the load were kept constant and the length l is the only design parameter to be estimated. Let us consider two input training patterns, say pattern two and pattern four, named because they correspond to a design length of two and four respectively. The rms values of the deflection vectors corresponding to patterns two and four are plotted against their corresponding design lengths as the dark squares at the extremeties of the lines in figure 4.

Now assume that we wish to estimate a design length for a deflection that is midway between that for pattern two and that for pattern four. We consider every element of the deflection vector to be midway between those for the two patterns.

If a look-up table is used, a length midway between that for pattern two and pattern four would be selected, that is three. This is shown as a linear interpolation by the dark square labelled *linear interpolation* in figure 4. It is obvious from the beam-truss-spring design equations (3) and the matrix, figure 2 that the deflection vector is not linearly related to the length so that this solution is not likely to be very good.

If the midway deflection vector is input to the trained neural network, a different design length is obtained, labelled *neural network output* and marked with an open box in figure 4. This can be shown to be superior to the linear result as follows. The design length obtained is used to determine the matrix **B**, figure 2. Equation (3) is used to compute a deflection vector. The rms of this deflection vector is computed and plotted as the *actual* value in figure 4. The result is close to the nonlinear interpolated result obtained from the neural network. This shows that the network produces a better interpolation than that obtained from the linear interpolation with the look-up table.

In practice it is always possible to determine the accuracy of the interpolation by using the interpolated design with analysis tools to see how well the resulting design meets its specification.

CONCLUSIONS

This paper showed that the split inversion neural network algorithm can provide accurate interpolation for non linear situations that would be difficult to obtain with a look-up table. The nonlinear relations need never be known. The network learns these from training with representative examples. The discretization of the parameters used in obtaining training samples and the nature of the underlying equations determine the accuracy of the interpolation. This approach was shown to be valuable for engineering design where expertise is often dependent on association with previous designs rather than by following a set of rules.

REFERENCES

1. McAulay A.D., “Neural networks as a new strategy for computing”, NAECON, May, 1988.

2. McAulay A.D., "Engineering Design Using Split Inversion Learning", IEEE First Annual International Conference on Neural Networks, June 1987.

3. McAulay A.D., "Optical Neural Network for Engineering Design", NAECON, May 1988.

4. Rumelhart D.E., Hinton G.E., and Williams R.J., "Learn ing Internal Representations by Error Propagation", in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, The MIT Press, Cambridge, 1986.

5. McAulay A.D., "Conjugate Gradients on Optical Crossbar Interconnected Multiprocessor", Journal of Parallel and Distributed Processing, Feb. 1989.