NETSYN: Neural Networks for Acquisition and Use of Synthesis Knowledge

James H. Garrett, Jr., Nenad Ivezic

EDRC 12-50-92

RESEARCH SUMMARY

NETSYN: Neural Networks for Acquisition and Use of Synthesis Knowledge

James H. Garrett, Jr., Assistant Professor, Department of Civil Engineering
Nenad Ivezic, Graduate Research Assistant (M.S. Candidate)

1. MISSION STATEMENT

The mission of this project is to investigate the advantages and disadvantages of using a neural network-based paradigm for performing synthesis knowledge acquisition and compare this approach to relevant symbolic approaches. The anticipated benefits of using a connectionist approach for synthesis knowledge acquisition (SKA) are:

- the synthesis knowledge, represented as a large collection of nonlinear, hyper-dimensional mappings between a current state of a design (a set of specification and design attributes and values) and several alternatives for the next design state (the next design attribute and value), will be able to provide synthesis advice for novel design state using a form of nonlinear interpolation between training cases; and
- the synthesis knowledge will be more efficiently represented, accessed and applied due to the distributed representation of knowledge over the network of interconnections.

The anticipated benefits of performing this research are:

- the advantages and disadvantages of a connectionist approach to SKA will be determined and compared to those of more traditional, symbolic approaches to machine learning;
- a set of guidelines will be developed to determine whether a connectionist approach, a symbolic approach, or some combination of approaches is appropriate for performing SKA; and
- in the process of doing future work (described later), the benefits of applying this form of machine learning within Ml synthesis tool [Gupta, 1990] will be investigated.

2. RESEARCH GOALS

The goals of this research are the following:

- to identify and assess candidate neural network models for acquiring and representing synthesis knowledge and for use of this acquired knowledge in novel design situations,
- to compare and contrast the performance of these candidate neural networks to that of the modified COBWEB symbolic learning algorithm used by Reich in developing M^2LTD [Reich, 1991], and
- to present directions for the future research of neural network models for learning and using design synthesis knowledge.

3. GENERAL APPROACH: NEURAL NETWORKS

A neural network is at the same time a computation, knowledge representation and knowledge acquisition paradigm. By its very nature, it integrates these three concepts. Neural networks compute by propagating activation through a very large network of interconnected simple processors *(neurons).* Each processor accepts a set of inputs from other processors and computes an output which is propagated to the connected processors. Programming, per se, does not exist. Networks are trained *(learning)* to recognize input patterns and produce appropriate output responses. Due to the distributed nature of the neural computing "machine,'* computations are robust and useful results can be obtained (and accuracy maintained) given partial or erroneous input, or when one or more individual neural processors fail.

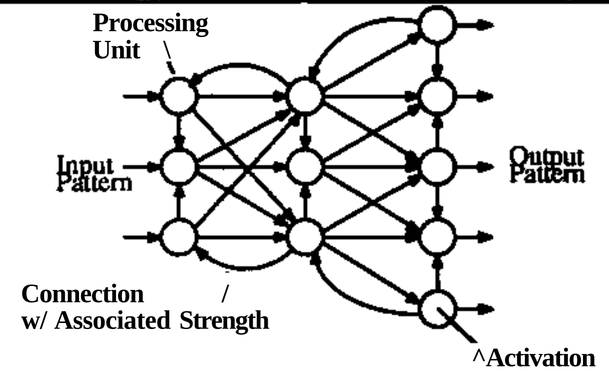


Figure 1. - Sample Neural Network

A neural system maps an input *vector* (defined in its broadest sense as a collection of elements) from one *space* to another space (e.g., mapping a current design state to the next state). The translation is not specified, but is learned. In learning, the explicit statement of what to learn is also not specified. A network is given a set of patterns *(inputs)* and their associated classifications *(outputs).* The network consists of several *layers* of interconnected neural elements (the network structure is patterned after its physiological equivalent). Just as electrochemical signals propagate through a set of neurons, in the neural computing model, a set of *activation* values pass through the network, transforming the inputs to the outputs.

Generally, each processing unit in the network maintains only one piece of dynamic information (its current *activation* level, a scalar), although, a more complex state is permitted. Neural networks compute by propagating activation through the network; each processor performs simple computations, taking its inputs and its activation value, computing a new activation, and propagating an output over its outgoing connections. Connections are directed paths in the network: the output from one processor passes to all the downstream processors. Associated with each connection is a measure of the strength of the connection—a *weight*—which is used to attenuate the signals.

Given an input pattern, a *correct* computation will produce the expected output pattern. For a given neural architecture (i.e., number of processors, connection pattern, activation functions), it is the set of weights of the connections between the processors that determines the output. Thus, the *program* is embodied within the topology and strengths of the interconnections. Changing what is computed would thus require changing the weights, the interconnection pattern or the activation functions. Manual programming (setting) of the weights is usually not performed. Rather the network is trained to exhibit "correct" behavior.

Training a network involves four steps: (1) present each case within the training set (a collection of input/output cases) to the network; (2) compare the output pattern produced by the network to the expected pattern; (3) backpropagate the error through the network; and (4) modify the weights of the connections according to the learning rule. This *learning* process is repeated until the behavior of the network is satisfactory (or the architecture is abandoned). After being trained on a large number of cases, a robust network will perform well on many cases which were not in the training set, i.e., it will *generalize.* Also, due to the structure of such networks, modifying individual weights, breaking a few connections, etc., will not have major impacts on the computed results. Thus a neural network is robust, will tolerate fuzzy or incorrect inputs, and will produce reasonable results when individual processors fail.

Thus, in summary, a neural network is a computing paradigm in that output patterns are computed for given input patterns through activation propagation. A neural network is a knowledge representation paradigm, where the topology and strength of connections are the means by which knowledge of association between various features of the input is represented. Most neural networks are also a knowledge acquisition paradigm in that a learning mechanism is provided to automatically acquire the set of weights, and in some cases, the topology as well.

4. PROGRESS

This project began on June 1,1991. To date, the work performed has been in two main areas: (1) defining the requirements for a neural network for acquiring synthesis knowledge; and (2) identifying and assessing candidate neural network models for acquiring design synthesis knowledge.

4.1 Identification of Requirements for Neural Network

Three requirements are identified as necessary ingredients of any system capable of capturing synthesis knowledge and using this knowledge in novel design situations (these requirements are based mainly on the previously completed research project in machine learning at EDRC by Yoram Reich [Reich, 1991]:

- the ability to capture many-to-many mapping between design specifications, design attributes and performance indices,
- the ability to recall or generate multiple design alternatives satisfying the particular combination of specifications, design attributes and performance indices (i.e., to acquire generative knowledge), and
- the ability to generalize from the individual design experiences - generating acceptable solutions in design situations not seen before.

4.2 Investigation of Neural Network Models for SKA

To date, several different neural network architectures have been investigated for acquiring and using synthesis knowledge. Simple feed-forward networks with analogue output units, where each output unit completely represents single design property, can be trained to map between an input space and an output space, as long as the mapping from input to output is unique. If the same input pattern can lead to different output patterns (as is the case in synthesis), these simple feedforward networks with analogue output units cannot be used. We immediately dismissed this simple, feed-forward network type as a viable approach to SKA. The reason is that this network architecture can not acquire generative knowledge (i.e., the second requirement).

One architecture that has shown promise is the Simple Recurrent Network (SRN) by Elman [Elman, 1988] (see Fig. 2). Processing in this type of network consists of the following sequence of events. At time t, the input units receive the first input in the sequence. Each input might be a single scalar value or a vector, depending on the nature of the problem. The context units are initially set to "don't know" value (0.5 for the activation function in the range between 0.0 and 1.0). The input units and context units both activate the hidden units. Then the hidden units feed forward to activate the output units. The hidden units also feed back to activate the context units. This constitutes the forward activation. Depending on the task, there may or may not be a learning phase in this time cycle. If so, the output is compared with a teacher input and backpropagation of error is used to incrementally adjust connection strengths. Recurrent connections from hidden to the context layer are fixed and are not subject to adjustment. At the next time step t+1, the above sequence is repeated. This time the context units contain values which are exactly the hidden unit values at previous time t. These context units provide the network with memory: they remember the previous internal state of network. Thus, the hidden units have the task of mapping both an external input and also the previous internal state to some desired output.

SRNs have been used to recognize, or predict, temporal sequences of signals. For example, in the domain of natural language processing, a SRN has been shown sequences of characters that are derived from some grammar. The network was able to predict the patterns of characters that follow some input sequence, and to recognize when some pattern was legally derived from the given grammar. Thus, the network, through learning, is capturing rules associated with some grammar that allows the networic to recognize and predict valid sequences derived from that grammar.

Because grammars can be used to both recognize and synthesize valid designs, these recurrent neural networks can be also used to acquire synthesis knowledge. Such a network is trained to select the next design attribute (and its value) given the design specification and the current state of the design. If the attribute values are discretized, the network acts like a finite state machine in that it identifies the states to which the design can progress and, in addition, provides advice about which attribute value would be the best to select based on past experience. Such a network, once trained,

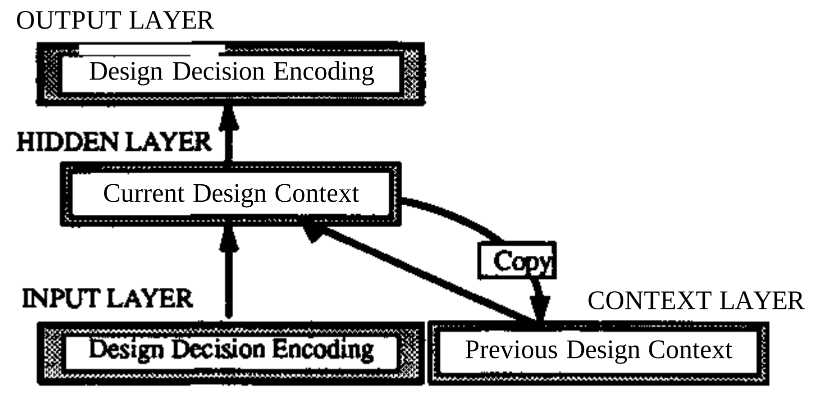


Figure 2. - A **Simple Recurrent Network**

would be used to iteratively move from one design state to the next; each subsequent design state recommended by the network is based on the input specifications and the design attributes previously selected.

For example, Fig. 3 shows the results of training a simple recurrent network on a set of local bridge design examples (see the Appendix at the end of this summary for a detailed description of this data set). The first three attributes are the specification attributes (total length to be spanned, number of lanes, and whether clearance is governing) and the last four are the design attributes (the deck position, the center span length, the relative length of center span to total length, and the bridge type used). The cases on which the network was trained are indicated by those paths through this tree ending in a shaded box. Fig. 3 illustrates the knowledge stored within the recurrent neural network using the following notation: 1) bold lines indicate that a high activation would occur for the attribute value, thus recommending its selection for the given design context; 2) normal lines indicate a medium activation would occur at the network output layer for this attribute value, thus recommending this attribute as a possible alternative value; and 3) dashed lines indicate that little or no activation would occur at the network output layer for this attribute value, thus recommending this attribute value not be considered for the current design context. The design context for a particular decision is the path of decisions made up to that point. Thus, while using this network, if the specifications were given as medium length (m), two lanes, and clearance not governing (n), the network would recommend that the decision about deck position be deck (d). If the user does select deck position = deck, the network would then recommend that the center span be of medium length (m).

The process of learning the synthesis knowledge and of recalling the acquired knowledge has been analyzed to some extent with respect to the following:

- *Time complexity of learning* was substantial even in the small problem used for our case-study: the time required to learn the underlying synthesis knowledge in the small database of IS bridge designs was dependent on the selected network architecture and network parameters, and it was on the order of minutes of wall clock time on a DEC 5100. However, other investigations point out that increasing the magnitude of the problem (i.e., number of design properties and training cases) does not necessarily result in a more difficult knowledge acquisition process. What seems to matter is the degree of correlation between the design attributes. An investigation remains to be performed with respect to the learning complexity for different sizes of training cases.
- *Time complexity of recall* in the trained network is insignificant for a realistic design process (on the order of fractions of a second for the above problem). The recall always takes constant time. We expect the recall time to scale up favorably with the size of the problem.

- *The performance of the trained network* (as illustrated in Fig. 3) has to be viewed with respect to following two metrics:

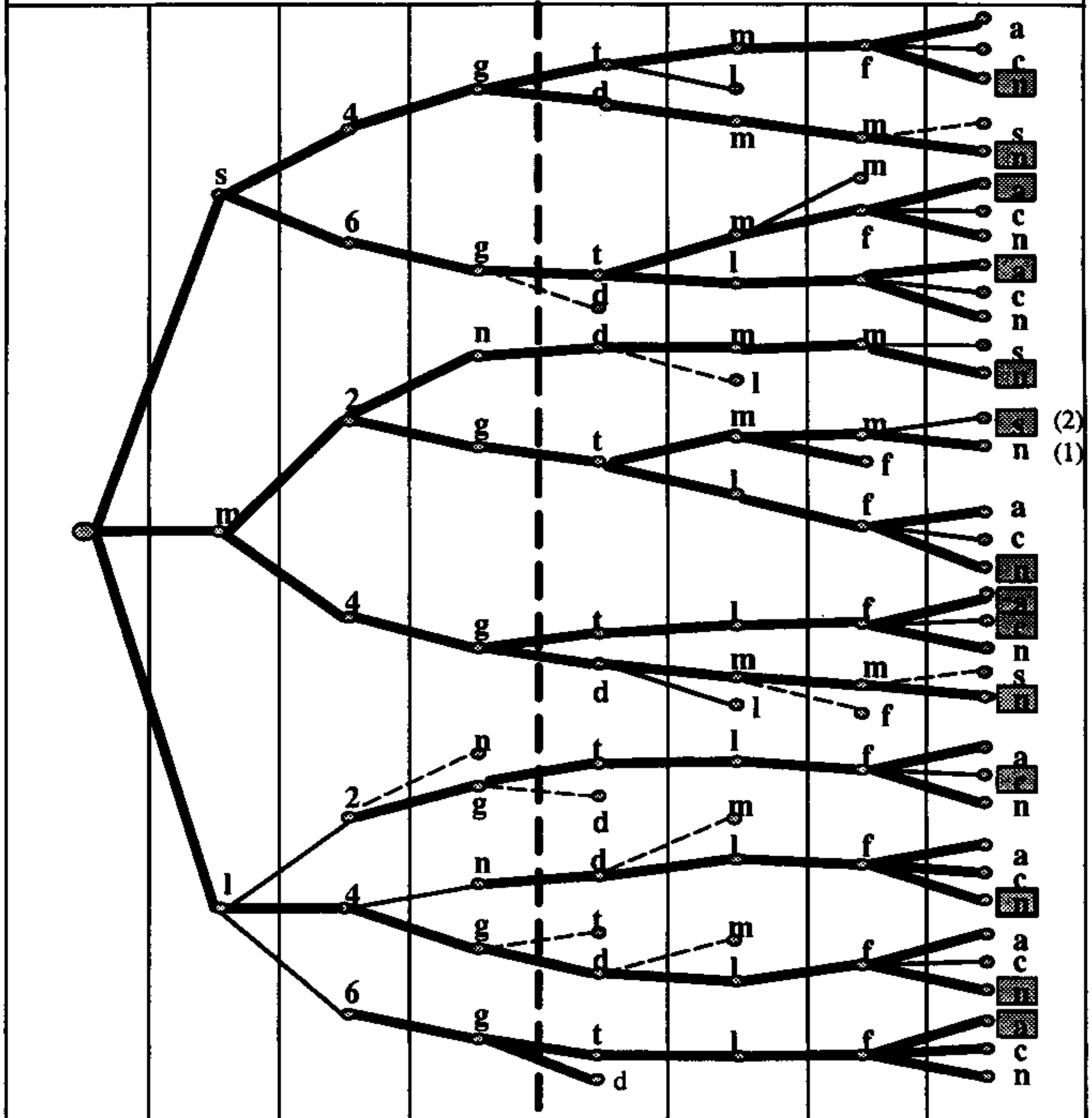
    - Accuracy - i.e., is the network capable of retrieving all training cases. In our case study we have witnessed satisfactory performance of this network type as it was able to retrieve the previously seen design cases (see Fig. 3 and Appendix).

    - Generalization - i.e., does the network predict designs that are distinct from those in the training set based on some generalization concept implicit in the network operation. The generalization mechanism in this particular network type is based on the correlation between the design property values seen in the training set Therefore, the network will predict with higher activations (confidence) those design property values that are correlated with a larger number of previously selected property values. As an example of this, in Figure 3, design (1) is predicted with greater confidence than (2), because in that particular design context the network has seen the continuous (n) bridges in a greater number of cases than the suspension (s) bridges.

The issue of generalization is a delicate one: it is the specific design domain for which the notion of generalization on the finite amount of synthesis knowledge has to be defined. For this reason it is necessary to include some domain-specific knowledge in the learning process which will cause the desired generalization process and still preserve the accuracy of the acquired knowledge. This remains a topic of further research.

You will note that the order in which the design attributes are considered is fixed and the values these attributes may take on are discrete. Future work is aimed at extending this simple recurrent network to handle various sequences of design attributes and to handle continuous attribute values.

Figure 3. - Results of Simple Recurrent Network applied to Pittsburgh Bridge Design Database



Property	Length	Lanes	Clearance	Deck–Pos	Span	Rel–len	Str–type
Values	small	2 lanes	not govern	through	medium	medium	arch
	medium	4 lanes	governing	deck	long	long	cantilever
	large	6 lanes					suspend.
							continuous
	Specifications			Attributes			

5. CURRENT APPROACH

The **current approach that** we **are** taking is based on the concept of associative memory. The approach alleviates the deficiency of **the** previous **approach** based on the Simple Recurrent Network as it does not require the design properties to be determined in **an a** priori fixed sequence. In general, any sequence of design property predictions may be **undertaken. This further blurres the** crisp distinction between the notion of design specifications, attributes, and performances. **Additionally, one is able to** use incomplete design specifications both during the training and during the recall phases.

5.1 Definitions and Concepts

We view **a** design as a collection *of property-value* pairs or, equivalently, collection of *design decisions;* in the following we will use these two notions interchangeable. A design process is then equivalent to the sequence of determinations of property values which ultimately lead to a complete design definition. Some property values are determined a priori, representing design specification values and, eventually, some preferred design attribute values. Therefore, all design decisions are made in some *design context* (i.e., a partial design description).

The sequence of design decisions, although relevant to the final outcome of the design, is not restricted in this approach. This leads to an *order-independent* approach to making design decisions (i.e., only the design decisions that have been made influence the value of the other decisions, irrespective of the order in which one has arrived at this design context). Additionally, we do not make a distinction between design specifications, design attributes, or performance indices, since they are all viewed uniformly as design properties. Fig. 4 shows two possible sequences of design decisions leading from an initial context to the final design context.

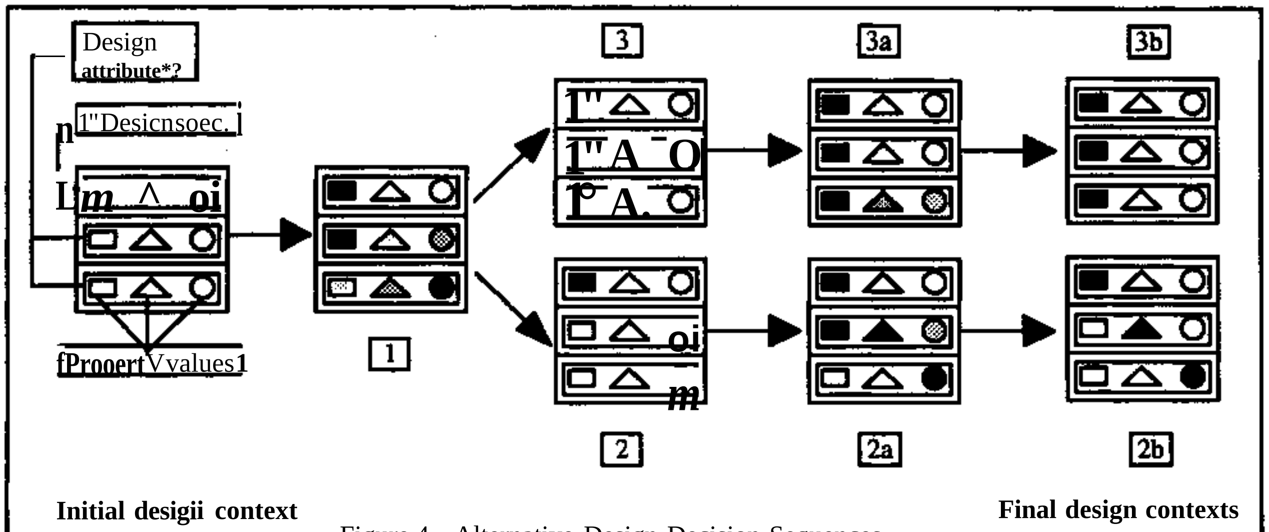


Figure 4 - Alternative Design Decision Sequences

To further elaborate **on** the meaning of adopted terminology, consider a representation of a simple design process in Fig. 4. On the left side of the figure, an initial design context is shown. To complete design, the values of three design properties need to be determined (single design specification and two design attributes). Each design property may take on one out of three distinct values (graphically represented by an rectangle, triangle, and an oval). In initial context, only design specification value is determined (denoted by solid shape).

The first task in the design process is to consider all the design properties which values have not been determined and to select one and assign its value. In the most general case we would like to consider every unresolved design property and decide which values are feasible for given design context. Then, based on some design strategy, we could select a property and its value for which we believe it will lead us to a good design. In the figure, completing this task gives the intermediate state of design context (1) and the new, more complete, design context (2) or (3). The intermediate task (1), represents consideration of undecided design properties. The different shades of fill patterns for value symbols depict possible state of our knowledge or belief of likelihood that each value may appear in given design context (the darker shades indicate greater likelihood). Alternative design contexts (2) and (3) are results of different design decisions.

A particular point is worth noting here: one usually has a number of alternative choices for making design decisions. The designer utilizes the subjective knowledge to determine likelihood with which each property value may appear in the given context; consequently, he or she decides which particular property value to determine next. We are specifically interested in learning what are the choices that we have for the design property value assignments and what is the likelihood that we should choose any particular property value in the given context. We acquire such a knowledge by using neural networks in the inductive fashion: we use existing records of previous designs to train the neural network to acquire the knowledge about the likelihood functions for each design property value assignment as a function of the design context

5.2 How Does Neural Network Learn Desired Likelihood Functions

We base our current approach on an associative memory concept found in certain types of neural network architectures. The associative memory concept can be thought of as generalization of the lookup table approach to information storage and retrieval. Compared to the usual lookup table form of memory, the associative memory has several characteristic properties:

- the process of storing information is adaptive: the connection weights are iteratively changed until the information is stored with sufficient accuracy in the network,
- associative memory networks have desirable property of noise-resistance, and
- associative memory networks can be used to implement content-addressable memory.

For the problems requiring noisy pattern recognition capability, where a certain level of noise in the input has to be tolerated or where an incomplete pattern description is supplied to the network to retrieve the complete information, the associative memory concept is an attractive approach. In addition, this concept has been used to learn the important features of specific network representations and the relationships between these features and the stored information. Similarly, it is possible to learn the underlying features of design descriptions and the influences each feature has on the selection of specific property-values.

Several different neural network architectures are capable of implementing the concept of associative memory (e.g., Hopfield network, Boltzman Machine). We have used in this study Multilayer Perceptron network architecture, a feed-forward neural network using the backpropagation learning rule. The basic difference between this approach and the feedforward network discussed in section 4.2 is in the mapping of the decisions of design property value assignments in the design problem to the output values of the network. By mapping each property value to a distinct set of output units, it is possibleto learn generative knowledge (as specified in learning requirements), which was obviously not possible when a single analogue unit was associated with the entire range of design property.

In spite of the similarity of design property value assignment to some other associative memory-learning tasks, such as noisy pattern recognition, there is a principal difference between our learning task and most of the other associative memory-learning tasks. While more traditional associative memory-learning tasks are concerned with learning and then recalling the closest matching pattern to a given input pattern, design property value assignment is concerned with learning an estimate of likelihood that any of the alternative property values will be selected as appropriate in the current design context.

This requirement on the learning mechanism illustrates a crucial capability of the Multilayer Perceptron network architecture. We have been using the notion of likelihood informally so far. However, it can be proved rigorously that the outputs of a trained multilayer perceptron approximate the a posteriori probability function when trained for multi-valued decision problem [Ruck et al., 1990]. In terms of our synthesis knowledge acquisition problem, this means that the adopted network architecture will allow us to capture statistically important properties of knowledge underlying the given training examples. Additionally, it has been shown that using standard Least Mean Squares (LMS) error criterion can lead to large errors in estimating probabilities (i.e., overestimating high probabilities and underestimating low probabilities). In our problem this would result in insensitivity of the learning algorithm to exceptional training cases and practically disregarding those designs that are not conforming to most often used set of rules. Therefore, the new error criterion similar to the Kullback-Leibler information measure was adopted, as proposed in [El-Jaroudi, 1990]. It has been shown that the use of this error function results in superior probability estimates compared to LMS criterion. We will continue to informally use the notion of likelihood although it is more correct to use instead the concept of posterior probability as pointed out

Figure 6 shows the idea of the algorithm adapted for our task, which consists of the following generic steps: (1) from the training set of design descriptions randomly select one design, (2) randomly select a subset of property-value pairs of the chosen design description (i.e., design context), (3) apply the design context to the input units of the network and propagate the signal forward, (4) compare the network output (estimate of the property-values likelihood function) to the correct value for this design instance description (particular value of the property taken from the selected design), (S) compute the error to propagate back through the network, (6) propagate the error back through the network and adjust network parameters.

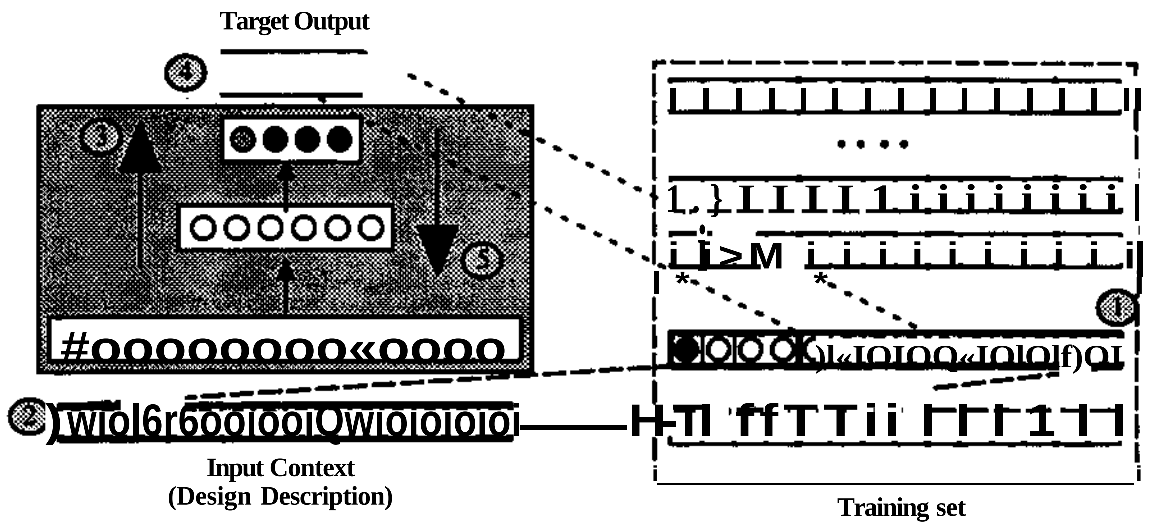


Figure 6 - Associative Memory-Based Network Architecture

5.3 An Associative Memory Approach for Synthesis Knowledge Acquisition and Use

The construction of an associative memory neural network is developed in a modular way: for each property of the design we can create a neural network structure which will learn the likelihood of this property assuming each of the possible values, given some design context In order to satisfy the requirement of learning many-to-many mappings, the neural network must be capable of learning the likelihood value of the property for any partial design description (i.e., design context). The likelihood function will indicate the likelihood that each property-value is appropriate for given design context (see Fig 5).

Figure 5 shows a particular feed-forward neural network architecture for capturing these likelihood functions, at which we are looking at presently. However, other architectures are likely to be candidates for this task, depending on their capabilities of learning specific synthesis knowledge. Investigation of these architectures is part of ongoing research.

By combining a number of modules that predict the property values for our design we can create one network that will predict all property values from the current design context at the same instance (Fig. 5). The benefit of this multiple property predictor is in providing a picture of confidence.of selecting the next property and fixing the corresponding value. A more informed design process follows from having a clear picture of which properties have values of high confidence as the design process is continued from the initial, to partially complete, to final (complete) design context.

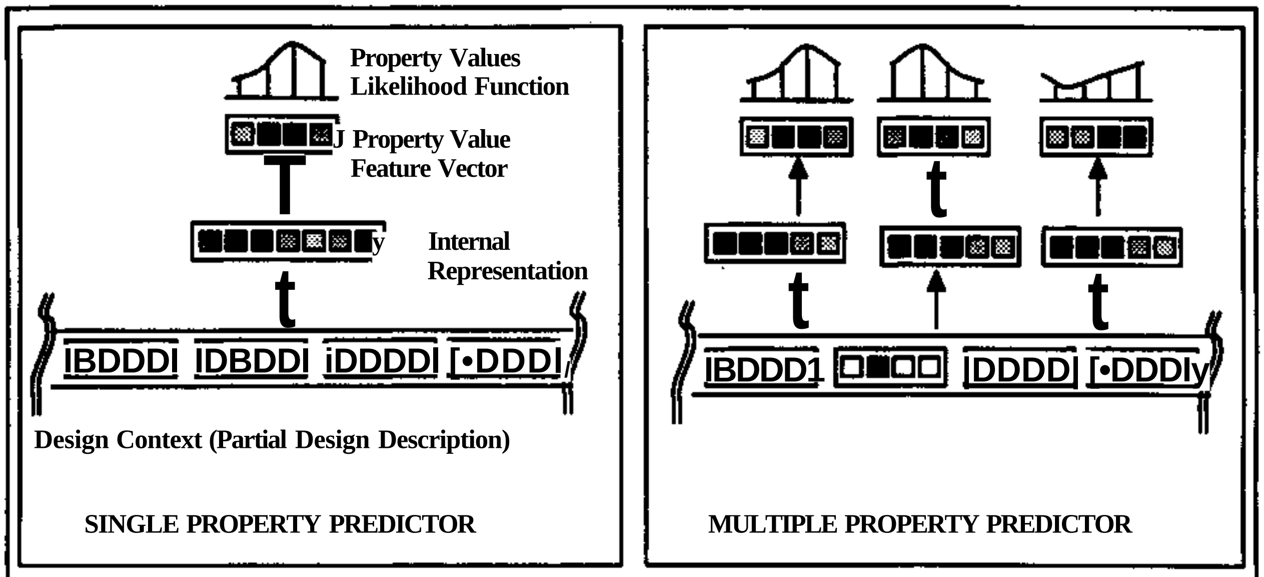

Property Values
Likelihood Function
Property Value
Feature Vector
Internal
Representation
Design Context (Partial Design Description)
SINGLE PROPERTY PREDICTOR
MULTIPLE PROPERTY PREDICTOR

Figure 5 - Neural Network for a Property Values Prediction

By providing a comprehensive view onto the state of the design in terms of the measure of likelihood that each property value is assigned, we see one of the attractive properties of this approach for using acquired knowledge in novel design situations. Future research will have as one of its focuses the integration of the described neural netwoik knowledge acquisition tool into the existing design synthesis tool Ml [Gupta, 1990].

6. FUTURE WORK

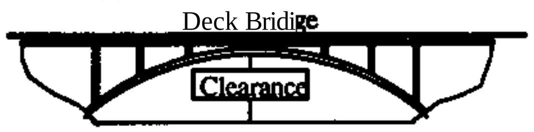
Based on our initial findings, we will perform the following tasks in the future:

- **We have designed and implemented the associative memory neural network model and are in the process of testing the results on relatively small design problems (Pittsburgh Bridge Database in Appendix, Concrete Placement Database).**
- **We will apply this neural network technique to the cable-stayed bridge data set used by Reich in his thesis and compare the results with those produced by ECOBWEB.**
- **We will apply these neural network techniques to MICON system to examine whether they can capture the synthesis knowledge and help designers in making more informed decisions at early and intermediate stages of the design process.**

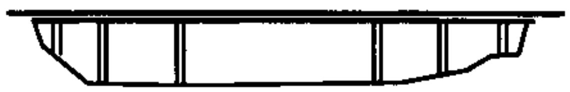
7. APPENDIX: Pittsburgh Bridge Database

Bridge Data:

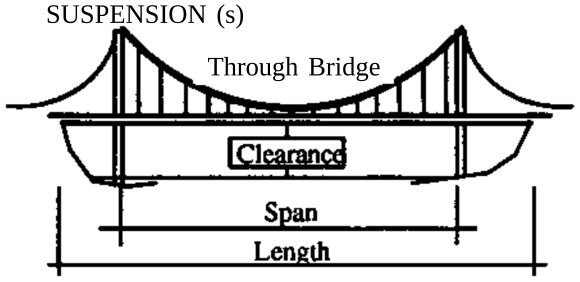
ARCH (a)



CONTINUOUS (n)

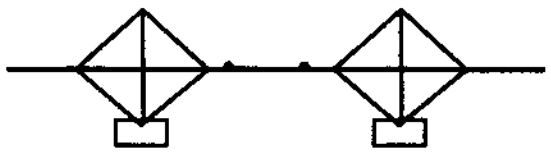


SUSPENSION (s)



Relative Length = Span/Length

CANTILEVER (c)



Training data:

Specifications										Design Attributes											
#	length			lanes				clr		deck		span			rel-1			type			
	s	m	1	1	2	4	6	n	y	t	d	s	m	1	s	rii	f	a	c	s	n
1.	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1
2.	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	1
3.	0	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	1	0	0
4.	0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0
5.	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1
6.	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1
7.	0	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1	0	0
8.	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1
9.	0	1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1
10.	0	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0
11.	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1
12.	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1
13.	1	0	0	0	0	0	1	0	1	1	0	0	1	0	0	0	1	1	0	0	0
14.	0	0	1	0	0	0	1	0	1	1	0	0	0	1	0	0	1	1	0	0	0
15.	1	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	1	0	0	0

8. REFERENCES

[Gupta, 1990] Gupta A. P. and D. Siewiorek, "Ml: a Small Computer System Synthesis Tool", Research Report; no. CMUCAD-90-8, Carnegie Mellon University.

[Reich, 1991] Reich, Y., "Building and Improving Design Systems: A Machine Learning Approach," Technical Report EDRC 02-16-91, Engineering Design Research Center, Carnegie Mellon University.

[Elman, 1988] Elman, J. L., "Finding Structure in Time", CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[El-Jaroudi, 1990] El-Jaroudi, A. and J. Makhoul, "A New Error Criterion For Posterior Probability Estimation With Neural Nets", UCNN International Joint Conference on Neural Networks, San Diego, 1990, pp 185-92.

[Ruck, 1990] Ruck, D. W., S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function", IEEE Transactions on Neural Networks, Vol. 1, No 4, December 1990, pp. 296-298.