TECHNICAL RESEARCH REPORT

Using Neural Networks to Generate Design Similarity Measures

by Sundar Balasubramanian, Jeffrey W. Herrmann

T.R. 99-38

Using Neural Networks to Generate Design Similarity Measures for Variant Fixture Planning

Sundar Balasubramanian and Jeffrey W. Herrmann

Department of Mechanical Engineering and Institute for Systems Research
University of Maryland
College Park, Maryland 20742

June 17, 1999

Abstract

This paper describes a neural network-based design similarity measure for a variant fixture planning approach. The goal is to retrieve, for a new product design, a useful fixture from a given set of existing designs and their fixtures. However, since calculating each fixture's feasibility and then determining the necessary modifications for infeasible fixtures would require too much effort, the approach searches quickly for the most promising fixtures. The proposed approach uses a design similarity measure to find existing designs that are likely to have useful fixtures. The use of neural networks to generate design similarity measures is explored. This paper describes the back-propagation algorithm for network learning and highlights some of the implementation details involved. The neural network-based design similarity measure is compared against other measures that are based on a single design attribute.

Keywords: Neural networks, variant fixture planning, design similarity.

1. Problem Statement

This paper describes a neural network-based design similarity measure for a variant fixture planning approach. The goal is to retrieve, for a new product design, a useful fixture from a given set of existing designs and their fixtures. However, since calculating each fixture's feasibility and then determining the necessary modifications for infeasible fixtures would require too much effort, the approach searches quickly for the most promising fixtures. The objective is to determine a design similarity measure that reflects fixture-usefulness. In mathematical terms, given a new design D, the neural network model is required to estimate usefulness of an existing fixture (of design D′) for design D. Input to the neural network model are the design attributes for both designs D and D′, while the output is given as follows:

$$\text{Output (Relative Metric)} = \frac{Q'}{Q} \tag{1.1}$$

where Q' is the usefulness metric for new design D with existing fixture (of design D') and Q is the usefulness metric for new design D with its best generatively designed fixture.

The best generatively designed fixtures provide the least maximum contact reaction force for an applied unit torque (clockwise or counter-clockwise). The reciprocal of this reaction force is called the torque resistance metric [Bro96], which is the measure of quality and usefulness of a fixture for a given design, in our discussion. If the maximum contact reaction force is smaller, the metric is larger. Therefore, for any design, the best generatively designed fixture yields the highest quality metric. This implies that the output for the neural network model lies in the range [0, 1].

In planar fixture synthesis, a design can be characterized by design attributes such as MID, Area and MVED. The Maximum Inter-vertex Distance (MID) for a part (or design) is the maximum length between any pair of vertices for the polygon that represents the 2D projection of a design. Similarly, the Area measure is the area covered by the polygon representing the design and the Maximum Vertex-Edge Distance (MVED) is the maximum perpendicular distance between any vertex and an edge for the polygon that represents the 2D projection of a design.

A design similarity measure based on any single design attribute like MID cannot represent fixture-usefulness accurately. However, a measure that uses a collection of these design attributes might together provide a better measure of fixture-usefulness. Note that parameters such as number of sides or number of edges do not reflect fixture-usefulness; for example – an arc in the 2D projection of a part can be approximated by chords, in which case the number of edges loses its significance as a design attribute. This paper describes a method that establishes a mapping between the design attributes mentioned above and fixture usefulness.

2. Background and Approach

2.1 Variant Fixture Planning

A variant fixture planning approach exploits existing knowledge by retrieving, for a new product design, a useful fixture from a given set (or database) of existing designs and their fixtures. It will enable a manufacturing firm to reuse dedicated fixtures by identifying an existing fixture that requires only a minor change to become an effective fixture for a new design. This will reduce the amount of time spent constructing fixtures. However, since calculating each fixture's feasibility and then determining the necessary modifications for infeasible fixtures would require too much effort, the approach searches quickly for the most promising fixtures.

As explained below, the proposed approach uses a design similarity measure to find existing designs that are likely to have useful fixtures. The design similarity measure allows the approach to identify the most promising fixtures (those that correspond to the most similar designs) and process only those in more detail.

A particular class of products and modular components has been considered. One face of the part rests on the supporting plane (a baseplate) and the fixture elements constrain all motion of the part in the supporting plane. Thus, only the 2D projection of any given design onto the supporting plane is considered for fixture planning. Only polygonal shapes are considered. In this setting, a fixture is a set of three locators (pins) and one clamp. Generative fixture planning approaches [Bro96, Zhu96] have been developed for this domain. However, because they create a unique fixture for each design, generative approaches do not support the need to reuse fixtures.

Given a new design *D*, and an existing design *D′*, we define the fixture-based design similarity measure h(D',D)

$$= e^{-Abs\left(\frac{Attr(D)}{Attr(D')} - 1\right)} \qquad \text{if Attr(D)} \geq 0.9*\text{Attr(D')}$$

$$= 0 \qquad \text{otherwise}$$

where Attr() represents any one of the design attributes MID, Area, and MVED.

Note that this design similarity measure follows the approach described in Balasubramanian *et al.* [Bal98] and is based on ideas from Herrmann and Singh [Her97]. Specifically, the design similarity measure reflects the underlying fixture usefulness because these attributes are related to fixture usefulness. In addition, the measure is not symmetric. However, these measures based on any single design attribute do not yield a consistent measure, as shown in Section 3. The remainder of this paper describes a neural network-based design similarity measure that utilizes these design attributes and establishes a mapping between the design attributes and fixture usefulness.

2.2 Introduction to Neural Networks

Artificial neural networks (ANNs) are computer simulations of biological neurons, composed of nonlinear computational elements operating in parallel. Neural networks consist of nodes (neurons) and synaptic connections that connect these nodes. Each connection is assigned a relative weight, also called connection strength, or synaptic strength. The output at each node depends on the threshold (bias or offset) specified and a transfer (activation) function.

In mathematical terms, a neural network model can be represented by the following constituent parameters [Mul90]:

1. A state variable s_i is associated with each node *i*.
2. A weight w_{ij} associated with a connection between a node *i* and, a node *j* that the node *i* is connected to, such that signals flow from *j* to *i*.
3. A bias v_i associated with each node *i*.
4. A transfer function $f_i(s_j, w_{ij}, v_i)$ for each node *i*, which determines the state of the node as a function of the summed output from all the nodes that are connected to node *i*, and its bias.

The state variable s_i of a node i is given by,

$$s_i = f_i\left(\sum_{j=1}^{J} w_{ij} s_j - v_i\right) \tag{1.2}$$

Note that transfer functions and bias terms are absent for the input nodes. The sigmoid function is the commonly used transfer function. Another popular activation function is the 'tanh' function. These functions are monotonic with a finite derivative. The sigmoid function is equivalent to the 'tanh' activation function if we apply a linear transformation $\tilde{\alpha} = \alpha/2$ to the input and a linear transformation $\tilde{f} = 2f - 1$ to the output. The sigmoid and 'tanh' functions are given as follows:

$$\text{Sigmoid function} = f(\alpha) = \frac{1}{(1+e^{-2\beta\alpha})} \tag{1.3}$$

$$\text{'tanh' function} = f(\alpha) = \frac{e^{\beta\alpha} - e^{-\beta\alpha}}{e^{\beta\alpha} + e^{-\beta\alpha}} \tag{1.4}$$

where β determines the steepness. Generally, β is set to unity, which results in a sigmoid transfer function of the form,

$$f(\alpha) = \frac{1}{(1+e^{-\alpha})} \tag{1.5}$$

If $y_i = f(h_i)$, where h_i is the summed input to a node i, $f'(h_i) = y_i(1-y_i)$. In other words, the sigmoid function lends itself to back-propagation since corrections to weights can be obtained in terms of the state values of the output nodes. This is discussed in detail in Section 2.3.

Some typical applications of neural networks involve pattern mapping, pattern completion and pattern classification. Interest in neural networks stem from the fact that these models are capable of performing complex tasks that are impossible with sequential models. Neural networks are particularly useful in problems where the logical structure or the input-output relation is poorly understood. Neural networks are capable of learning and generalizing from examples in the absence of explicit rules or an analytical structure.

Classification of Neural Network Models

A neural network is initialized with a random set of weights. Adjustment of the connection weights to improve a predefined performance measure of a neural network is called *learning*. There are two types of learning methods: supervised and unsupervised. In the case of supervised learning, a set of training input vectors and their associated output vectors are presented to adjust the weights in the network. In unsupervised learning, no output vectors are specified. Strategies such as those based on *penalty* and *reward*, and genetic algorithms are employed to adjust the synaptic strengths.

Neural nets can also be classified into feed-forward and feed-back networks, on the basis of direction in which signals flow. In a feed-forward network, signals flow in only one direction, from the input layer through the intermediate hidden layers to the output layer. Neurons in the same layer do not communicate with each other. In a feedback network, signals can flow from the output of any node to the input of any node.

Neural nets can have binary inputs or continuous valued inputs. A neural network model that consists of an input neuron layer that feeds directly into an output layer is termed as a *simple perceptron.* However, neural network models can possess inner, or *hidden*, neuron layers that intervene between the input and output layers. Such models are called *multi-layered* networks. The best-studied class of layered neural networks is the feed-forward network. Within layered networks, there are fully-connected and partially connected networks. In a fully-connected network, all the nodes in each layer are connected to all the nodes in the previous layer, while this criterion does not hold for a partially connected neural network topology. See [Lip87] for a neural net taxonomy.

Multi-layered Networks

Multi-layered are networks with input, output, and inner (or hidden) neuron layers that intervene between the input and output layers. The input-output relation defines a mapping, and the neural network provides a representation of this mapping. The number of hidden layers and number of nodes in each hidden layer depend on the complexity of the problem, and to a large extent vary from problem to problem. Increasing the number of hidden layers increases the complexity of a neural network, and may or may not enhance the performance of the network. In most cases, more nodes in a hidden layer result in a better network performance but lead to a longer training time. Based on previous experience, one or two hidden layers provide a better performance [Hua99], while not requiring extensive training time. Sensitivity analysis can be performed to obtain an optimal model structure, by varying the number of hidden layers and the number of nodes per layer and evaluating performance for each of these alternative models.

Three-layered Feed-forward Network

A three-layer feed-forward network consists of an input layer, an output layer and a hidden layer, resulting in two layers of weights – one connecting the input to the hidden layer and the other connecting the hidden layer to the output layer. Figure 1 illustrates a three-layered feed-forward implementation architecture for evaluating design similarity measures with MID, Area and MVED parameters for both the existing and the new design as inputs and the relative metric as the output. (Note that not all weights are shown).

The states of nodes in the various layers are as follows:

$$\text{Hidden layer: } \bar{y}_j = f(\bar{h}_j) \qquad \bar{h}_j = \sum_{k=1}^{K} \bar{w}_{jk} x_k \qquad \text{for } j = 1, \ldots, J \tag{1.6}$$

$$\text{Output layer: } y_i = f(h_i) \qquad h_i = \sum_{j=1}^{J} w_{ij} \bar{y}_j \qquad \text{for } i = 1, \ldots, I \tag{1.7}$$

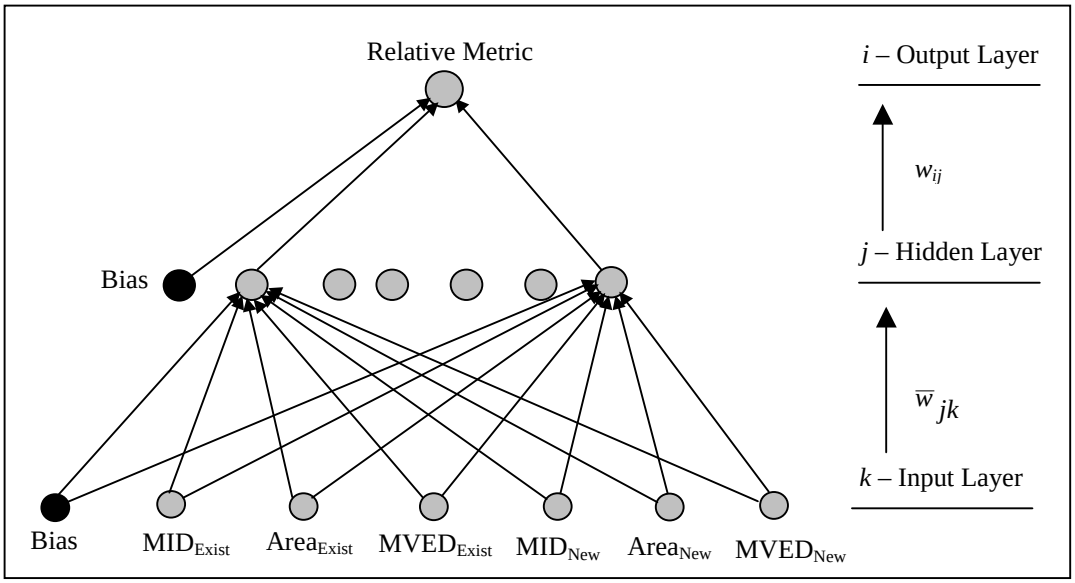

Figure 1. A three-layered feed-forward network

Note that the subscript *i* refers to one of the I output nodes, the subscript *j* refers to one of the J nodes in the hidden layer (including the bias node), and the subscript *k* refers to one of the K input nodes (including the bias node).

The topology presented in Figure 1 is equivalent to the description in Section 2.2. The bias v_i associated with each node *i* is eliminated and instead, an additional bias node is added to the input and hidden layers. From Equation (1.2),

$$s_i = f_i\left(\sum_{j=1}^{J} w_{ij}s_j - v_i\right) = f_i\left(\sum_{j=1}^{J} w_{ij}s_j - w_{i(j+1)}\right) = f_i\left(\sum_{j=1}^{J+1} w_{ij}s_j\right) \tag{1.8}$$

where $w_{i(j+1)}$ is the weight associated with the connection between the bias node *j+1* (that feeds into node *i*) and node *i*. The bias nodes always have a value of 1. In other words, $\sigma_0 = \bar{y}_0 = 1$. The bias nodes act in a manner equivalent to the intercept term in regression models. In the example shown in Figure 1, there are seven nodes (which includes the bias node) in the input and hidden layers. The output layer has only one node.

2.3 The Back-Propagation Algorithm

Learning is accomplished through an adaptive procedure, known as a *learning rule* or *algorithm*. Learning algorithms indicate how weights should be incrementally adapted to improve a predefined performance measure. Learning can be viewed as a search in a multidimensional weight space for a solution, which gradually optimizes a predefined objective function [Has95]. Back-propagation is the most extensively used training method. The back-propagation algorithm is an iterative gradient method based algorithm developed to introduce synaptic corrections (weight adjustments) by minimizing the sum of squared error (objective function).

Back-Propagation in a Three-layered Feed-forward Network

The aim is to choose weights such that the output deviation function is minimized. The output deviation function (sum of squared error) is given as follows:

$$\mathbf{D} = \frac{1}{2}\sum_{i=1}^{\mathrm{I}}[d_i - y_i]^2 \tag{1.9}$$

where i represents an output node, y_i = Actual network output; d_i = Desired output.

Adopting the method of steepest descent, we can determine corrections to weights as follows [Mul90] (considering connections for the output nodes):

$$w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij} \tag{1.10}$$

$$\delta w_{ij} = -\varepsilon\frac{\partial D}{\partial w_{ij}} = \varepsilon[d_i - y_i]f'(h_i)\frac{\partial h_i}{\partial w_{ij}} \tag{1.11}$$

where ε = learning rate.

For the sigmoidal transfer function [given $y_i = f(h_i)$],

$$f'(h_i) = y_i(1-y_i) \tag{1.12}$$

$$\frac{\partial h_i}{\partial w_{ij}} = \bar{y}_j \tag{1.13}$$

Substituting (1.12) and (1.13) in (1.11), we get,

$$\delta w_{ij} = \varepsilon[d_i - y_i]y_i(1-y_i)\bar{y}_j = \varepsilon\ \ \Delta_i\bar{y}_j \tag{1.14}$$

where $\Delta_i = [d_i - y_i]y_i(1-y_i)$

For weights associated with synaptic connections between the input and hidden layer,

$$\bar{w}_{jk}(t+1) = \bar{w}_{jk}(t) + \delta\bar{w}_{jk} \tag{1.15}$$

$$\delta\bar{w}_{jk} = -\varepsilon\frac{\partial D}{\partial\bar{w}_{jk}} = \varepsilon\sum_{i=1}^{\mathrm{I}}[d_i - y_i]f'(h_i)\frac{\partial h_i}{\partial\bar{y}_j}\frac{\partial\bar{y}_j}{\partial\bar{w}_{jk}} \tag{1.16}$$

$$\delta\bar{w}_{jk} = -\varepsilon\frac{\partial D}{\partial\bar{w}_{jk}} = \varepsilon\sum_{i=1}^{\mathrm{I}}[d_i - y_i]f'(h_i)w_{ij}f'(\bar{h}_j)\frac{\partial\bar{h}_j}{\partial\bar{w}_{jk}} \tag{1.17}$$

For a sigmoid transfer function,

$$f'(\bar{h}_j) = \bar{y}_j(1-\bar{y}_j) \tag{1.18}$$

$$\frac{\partial \overline{h}_j}{\partial \overline{w}_{jk}} = x_k \tag{1.19}$$

Substituting (1.18) and (1.19) in (1.17), we get:

$$\delta \overline{w}_{jk} = \varepsilon \sum_{i=1}^{\mathrm{I}} [d_i - y_i] y_i (1 - y_i) w_{ij} \overline{y}_j (1 - \overline{y}_j) x_k \tag{1.20}$$

$$\delta \overline{w}_{jk} = \varepsilon \overline{y}_j (1 - \overline{y}_j) x_k \sum_{i=1}^{\mathrm{I}} [d_i - y_i] y_i (1 - y_i) w_{ij} \tag{1.21}$$

$$\delta \overline{w}_{jk} = \varepsilon x_k \overline{\Delta}_j \qquad \text{where } \overline{\Delta}_j = \overline{y}_j (1 - \overline{y}_j) \sum_{i=1}^{\mathrm{I}} \Delta_i w_{ij} \tag{1.22}$$

Convergence can be accelerated by adding a momentum term as follows [Lip87]:

$$w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij} + \eta (w_{ij}(t) - w_{ij}(t-1)) \tag{1.23}$$

$$\overline{w}_{jk}(t+1) = \overline{w}_{jk}(t) + \delta \overline{w}_{jk} + \eta (\overline{w}_{jk}(t) - \overline{w}_{jk}(t-1)) \tag{1.24}$$

where η = momentum factor and $0 < \eta < 1$.

In the region of weight space for which the error surface has relatively low curvature, it can be shown [Bis95] that the momentum term aids in increasing the learning rate ε to an effective learning rate of $\varepsilon/(1-\eta)$.

This recursive method of back-propagation can be further extended for networks with more than one hidden layer. The values for the parameter constants such as ε and η can be altered to either improve performance or to reduce computational effort. This is further discussed in Section 2.6. There is no general criterion to determine these constants. The optimal values depend on the problem at hand.

Back-Propagation in a Four-layered Feed-forward Network

Here, the subscript *j* refers to one of the J nodes in the second hidden layer, subscript *k* refers to one of the K nodes in the first hidden layer, and the subscript *l* refers to one of the L input nodes.

$$1^{\text{st}} \text{ Hidden layer:} \quad \hat{y}_k = f(\hat{h}_k) \qquad \hat{h}_k = \sum_{l=1}^{\mathrm{L}} \hat{w}_{kl} x_l \qquad \text{for } k = 1, \ldots, \mathrm{K} \tag{1.25}$$

$$2^{\text{nd}} \text{ Hidden layer:} \quad \overline{y}_j = f(\overline{h}_j) \qquad \overline{h}_j = \sum_{k=1}^{\mathrm{K}} \overline{w}_{jk} \hat{y}_k \qquad \text{for } j = 1, \ldots, \mathrm{J} \tag{1.26}$$

$$\text{Output layer:} \quad y_i = f(h_i) \qquad h_i = \sum_{j=1}^{\mathrm{J}} w_{ij} \overline{y}_j \qquad \text{for } i = 1, \ldots, \mathrm{I} \tag{1.27}$$

For weights associated with synaptic connections between the input and first hidden layer,

$$\hat{w}_{kl}(t+1) = \hat{w}_{kl}(t) + \delta\hat{w}_{kl} \tag{1.28}$$

$$\delta\hat{w}_{kl} = \varepsilon\hat{\Delta}_k x_l \qquad \text{where } \hat{\Delta}_k = \hat{y}_k(1-\hat{y}_k)\sum_{j=1}^{J}\overline{\Delta}_j w_{jk} \tag{1.29}$$

The Algorithm [Lip87]

1. Initialize all weights to random values. Weight initialization is discussed in Section 2.5.
2. Prepare a data set of input vectors $\mathbf{x_0}$, $\mathbf{x_1}$, …… $\mathbf{x_m}$, with their associated output vectors $\mathbf{d_0}$, $\mathbf{d_1}$, …… $\mathbf{d_m}$. In our case, we have 729 sets of input/output pairs. Of this, we reserve 25% (183) for testing (validating) and use the other 546 for training.
3. Present the first input vector for training. Evaluate actual output. Starting at the output node, work back to the hidden layers and the input layer to adjust the weights, as described in Section 2.3. Then present second input vector, adjust weights, and repeat for all 546 examples in the training set.
4. At the end of this training cycle, test the network with the testing data set (183 sets of input/output pairs). Evaluate Mean Absolute Deviation (Other measures can also be used). If MAD ceases to improve (See Section 2.7), stop training; else repeat steps 3 and 4 for another training cycle.

$$\text{Mean Absolute Deviation (MAD)} = \frac{\sum_{n=1}^{N}\left|Y_n - \hat{Y}_n\right|}{N} \tag{1.30}$$

where N = Testing sample size (N = 183)

$\hat{Y}_n$ = Predicted (actual) output of for sample *n*

Y_n = Desired output of sample *n*

Note that the algorithm described above represents a sequential learning approach. An alternative to this technique is the batch mode where weights are adjusted at the end of each training cycle as against weight correction after each presentation of an input vector (as seen in the sequential approach). An advantage of the sequential approach is its relative potential to escape from local minima.

2.4 Pre-processing

Neural networks in principle can map raw input values directly into require final output values. However, in most situations neural networks do not yield satisfactory results without having the input data pre-processed. In many practical applications, the choice of pre-processing appears to be one of the most significant factors in determining the performance of the neural network model [Bis95]. It is often necessary to transform the input data into some new representation before presenting them to the network.

Each input vector has to be pre-processed before it is passed on to the network. Sometimes, post-processing of the output values may also be essential. The output values

must be post-processed using a post-processing transformation. In the training phase, the target values must be inverse-transformed to obtain the output values for training.

Standardization

Standardization is one of the most common techniques in pre-processing input data. It is essentially a linear scaling of the input variables so that different variables having values which differ by orders of magnitude can be presented in a manner where the relative importance of the different input variables in determining the output values is maintained. For example, in our implementation, the MID and the MVED parameters for both the existing and the new part fall within a range of 40-150, while the Area parameter falls within a range of 900-9700. The aim is to transform every input variable so that every variable in the input vector is of the same order of magnitude.

Suppose $\bar{x}_i$ is the sample mean of an input variable x_k, $\bar{\sigma}_k^2$ is the sample variance with respect to the training set of input data.

$$\bar{x}_k = \frac{1}{M}\sum_{m=1}^{M} x_k^m \tag{1.31}$$

$$\sigma_k^2 = \frac{1}{M-1}\sum_{m=1}^{M} (x_k^m - \bar{x}_k)^2 \tag{1.32}$$

where x_k^m (m = 1,…,M) is the value of variable x_k in the mth input vector of the training set, and M is the total number of input vectors in the training set.

Each instance of the input variable is transformed as follows:

$$\hat{x}_k^m = \frac{x_k^k - \bar{x}_k}{\sigma_k} \tag{1.33}$$

The re-scaled variable $\hat{x}_k$, associated with x_k in the training set, has a zero mean and unit standard deviation. Note that each variable is transformed independently. This ensures that all of the input and output variables are of the order unity, and hence, the weights would also be restricted to order unity. For the first training iteration, weights can be randomly initialized restricting them to order unity. In the absence of pre-processing (and, in some cases, post-processing), weights may have differing orders of magnitude.

2.5 Weight Initialization

Most of the initialization techniques aim at setting weights to randomly chosen small values. This is required in order to avoid symmetries in the network. The sigmoid function outputs a value very close to zero or one (in other words, the function is saturated) if it receives an input that lies outside [-3,3]. Therefore, the initial weights should be small so that the sigmoid transfer functions are not driven into their saturation regions. However, if weights are too small, the sigmoidal functions will be approximately

linear. As a result, nonlinearity in the network will be lost and training will be slower. It is desirable to maintain the summed inputs to sigmoidal functions within order unity.

Given that the inputs are standardized (rescaled so as to have a zero mean and unit standard deviation), initial weights can be generated from a Gaussian distribution with a zero mean and a variance $\sigma^2 \alpha 1/d$, where d is the number of nodes in the layer where the connections originate [Bis95]. Suppose there are seven input neurons feeding into the first hidden layer and thirteen nodes in the hidden layer feeding into the output layer. Weights associated with arcs that connect nodes in the input layer to nodes in the hidden layer are generated with a standard deviation $\sigma \alpha 1/\sqrt{7}$, while weights associated with arcs that connect nodes in the hidden layer to nodes in the output layer are generated with a standard deviation $\sigma \alpha 1/\sqrt{13}$. In our implementation, $\sigma = 1/\sqrt{d}$.

2.6 Adaptive Learning

Setting a value for the learning rate is essentially a trade-off between speed of convergence and the ability to closely approximate the gradient path. When ε is small, convergence will be slow due to a large number of update steps needed to reach a local minima. With a large ε, convergence is fast initially, but will induce oscillations and the error function will not reach a minimum [Lin97]. One possible strategy [Has95] is to use a large step size when the iteration is far from a minimum and a decreasing step size when the iteration approaches a minimum. However, this induces oscillations around the region where there is a decrement in the learning rate. Therefore, in practice, a constant value for the learning rate is used as this generally leads to better results even though the guarantee of convergence is lost [Bis95].

Another heuristic [Has95] to accelerate learning is to use learning rates, specific for each node, which are proportional to the number of nodes that feed in. If there are seven nodes in the input layer and thirteen nodes in the hidden layer, the learning rate for each node in the hidden layer is inversely proportional to seven, while that for each node in the output layer is inversely proportional to thirteen.

2.7 Cross-Validation

When the performance of a network is determined by evaluating the error function with respect to an independent testing (validating) data, it is observed that the validation error decreases monotonically to a minimum but then starts to increase. In general, multiple local minima may also exist in the validation error curves. This is attributed to excessive training, when training with noisy data, leading to over-fitting. Therefore, training is continued as long as performance on the validation set keeps improving. The set of weights that yields the least MAD is retained. This method of partial training may lead to a better generalization.

3. Preliminary Results

This section presents some of the preliminary results and observations for the proposed model for design similarity prediction. As mentioned earlier, there is no methodology to select the number of hidden layers, number of nodes in each hidden layer, connectivity,

and a learning algorithm. This necessitates constructing networks with different values for the model parameters to find the optimal configuration.

For training and testing, we created 27 designs and generated each design's best fixture using *FixtureNet*. Then, we evaluated the usefulness of each fixture on itself and all other designs. This yielded 729 (27^2) pairs of designs, each with a relative usefulness metric. Note that, a fixture yields a relative usefulness metric of 1 with its own design.

Feed-forward networks with sigmoid activation functions for hidden and output layers are considered. Learning is by the Least Mean Square based Back Propagation Algorithm described in Section 2.3. The learning rate for a network is represented by ε'. However, the learning rate for each node is determined by the heuristic [Has95] described in Section 2.6. For example, for a three-layered network with number of hidden nodes = 13, $\varepsilon' = 0.07$, learning rate for the hidden layer = $\varepsilon'/7 = 0.07/7 = 0.01$, while the learning rate for the output layer = $\varepsilon'/13 = 0.07/13 = 0.0054$.

When ε' is set at a relatively small value, the algorithm gets caught in local minima and does not approach the global minimum. Oscillations are observed as the learning rate is increased (See Figure 2). An appropriate range of learning rate values is determined and the network is tested for different values of learning rate in this range. In our implementation, the momentum factor is maintained at $\eta = 0.9$ for all experiments.

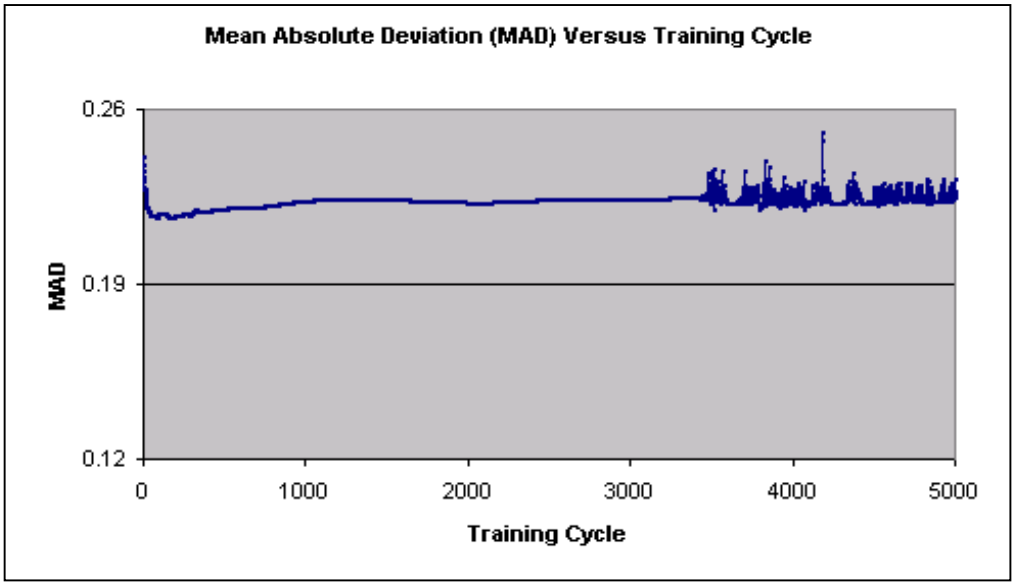

Figure 2. Oscillations are observed when $\varepsilon' = 0.5$ for a three-layered network with seven nodes in the hidden layer.

Experiments are conducted for networks with one and two hidden layers, while the number of hidden nodes is set at either seven or thirteen (including the bias node). Neglecting the bias nodes in the hidden and input layers, a network with seven nodes in the hidden layer has equal number of hidden nodes and input nodes, while a network with thirteen nodes in the hidden layer has two hidden nodes for every input node. The results

are tabulated in Table 1. A configuration is specified by the number of hidden layers n_1, number of hidden nodes n_2, and a learning rate ε'. $f(n_1, n_2, \varepsilon')$ is the minimum MAD observed in 5000 training cycles.

ε′	f(1,7, ε′)	f(1,13, ε′)	f(2,7, ε′)	f(2,13, ε′)
0.01		0.217768		
0.04	0.219189	0.211462	0.215588	0.210713
0.05		0.214961	0.213404	0.210757
0.07	0.209749	0.21449	0.209364	0.216243
0.09	0.210894	0.217342	**0.206576**	0.207592
0.1	0.216591	0.218244	0.210914	0.21443
0.2	0.208051			0.21481
0.3	0.207501		0.215314	0.207629
0.5	0.216103			

Table 1. Network performance (MAD) Versus Network Structure

A network with two layers and seven nodes in the hidden layer is identified as the optimal network model and the associated minimum MAD is 0.206576. However, these experiments are not exhaustive. By further varying learning rates and momentum parameters, we might arrive at a better network structure. Note that there is little or no improvement with increase in the number of hidden layers or the number of nodes within each hidden layer.

A comparison of the neural network model performance against the performance of other measures based on MID, Area and MVED (obtained from the approach described in Section 2.1) is presented below. Table 2 shows the correlation coefficient between the design similarity measure against the desired relative metric for measures based on MID, Area, MVED and the Neural net. The Neural network-based design similarity measure performs better than the other models and exhibits a high correlation.

	MID	Area	MVED	Neural Network
Correlation Coefficient	0.3593	0.3432	0.1558	0.6001

Table 2. Correlation Coefficients for various design similarity models

For ten new designs, we compared the best existing fixture identified by different similarity measures to the best generatively designed fixture (See Table 3). An observation of the relative metric (ratio of usefulness for the existing fixture of the most similar part to the usefulness of the generatively designed fixture) of these identified fixtures indicates that the neural network model, on an average, identifies a fixture with a higher usefulness metric. In other words, the average relative metric for the best existing fixture identified by the different measures is highest for fixtures identified by the neural network model. Also, the design similarity measure based on the neural network model identifies a better fixture in nine out of the ten cases.

	Usefulness Metric for Most Similar Part					Relative Metric of Most Similar Part			
New Part Number	MID (Metric)	Area (Metric)	MVED	Neural Net	Generative Fixture Metric	MID (Relative Metric)	Area (Relative Metric)	MVED (Relative Metric)	Neural Net (Relative Metric)
1	7.37	0	11.399	14.629	69.292	0.1064	0	0.1645	0.2111
2	20.975	0	21.617	22.344	69.717	0.3009	0	0.3101	0.3205
3	0	0	0	7.832	13.229	0	0	0	0.592
4	0	0	0	16.148	33.773	0	0	0	0.4781
5	0	15.595	15.584	28.454	33.549	0	0.4648	0.4645	0.8481
6	0	0	0	0.695	23.366	0	0	0	0.0297
7	3.768	0	0	0	36.504	0.1032	0	0	0
8	13.977	13.977	13.977	16.726	16.726	0.8356	0.8356	0.8356	1
9	15.426	0	21.127	21.127	28.253	0.546	0	0.7478	0.7478
10	0	0	0	7.36	8.334	0	0	0	0.8831
Average						0.1892	0.13	0.2523	0.5111

Table 3. Performance of the neural network-based design similarity measure versus measures based on MID, Area, and MVED in identifying the best available fixture.

4. Conclusions

This paper presented a variant fixture planning approach that uses a fixture-based design similarity measure to find promising fixtures quickly. Thus, this approach avoids checking the feasibility of each existing fixture. For each promising fixture, the approach calculates a more precise usefulness metric that describes how well the fixture can hold the new design.

Design attributes that reflect fixture usefulness were identified. Design similarity measures based on any of these design attributes have been discussed. The use of neural networks to represent the mapping between design attributes and fixture usefulness has been explored. Compared to other measures based on design attributes that reflect fixture usefulness, the neural network-based design similarity measure exhibits better consistency and, on average, identifies an existing fixture with a higher usefulness metric.

Future work will continue to integrate the variant fixture planner with a generative planner that creates a preliminary process plan to build a hybrid variant-generative process planner. This fixture planning approach also could be applied to fixture planning in other domains.

5. References

[Bal98] S. Balasubramanian, A. Elinson, J. Herrmann, and D. Nau, "Fixture-Based Usefulness Measures for Hybrid Process Planning," *Proceedings of the1998 ASME Design Engineering Technical Conference*, Atlanta, September 1998.

[Bis95] Bishop, Christopher M., "Neural Networks for Pattern Recognition," *Oxford University Press Inc.*, New York, 1995.

[Bro96] Brost, R.C. and Goldberg, K.Y., A Complete Algorithm for Designing Planar Fixtures using Modular Components, *IEEE Transactions on Robotics and Automation*, Volume 12, Number 1, February 1996.

[Has95] Hassoun, Mohamad H., "Fundamentals of Artificial Neural Networks," *The MIT Press*, Cambridge, Massachussetts, 1995.

[Her97] Herrmann, Jeffrey W., and Gurdip Singh, "Design similarity measures for process planning and design evaluation," *Technical Report 97-74*, Institute for Systems Research, University of Maryland, College Park, 1997.

[Hua99] Huang, C. –L., Huang, Y. –H., Chang, T. –Y., Chang, S. –H., Chung, C. –H., Huang, D. –T. and Li, R. –K., "The Construction of Production Performance Prediction System for Semiconductor Manufacturing with Artificial Neural Networks," *International Journal of Production Research*, Vol. 37, No. 6, pages 1387-1402, 1999.

[Lin97] Lin, Z. –C., and Huang, J. –C., "The Applications of Neural Networks in Fixture Planning by Pattern Classification," *Journal of Intelligent Manufacturing*, Vol. 8, pages 307-322, 1997.

[Lip87] Lippmann, Richard P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pages 4-22, April, 1987.

[Mul90] Muller, B. and Reinhardt, J., "Physics of Neural Networks, Neural Networks: An Introduction," *Springer-Verlag*, 1990.

[Phi94] Philipoom, P. R. and Rees, L. P., Wiegmann, L., "Using Neural Networks to Determine Internally-Set Due-Date Assignments for Shop Scheduling," *Decision Sciences*, Vol. 25, pages 825-851, 1994.

[Zhu96] Zhuang, Y. and Goldberg, K. Y., "On the existence of Solutions in Modular Fixturing", International Journal of Production Research, Volume 15, Number 6, pages 646-656, 1996.