Project Modeling for Discrete Event Simulation by Neural Networks and Fuzzy Associative Memories

O. E. Velipasaoglu  
*Purdue University School of Electrical Engineering*

O. K. Ersoy  
*Purdue University School of Electrical Engineering*

T. Nienhaus  
*Purdue University School of Electrical Engineering*

Follow this and additional works at: [http://docs.lib.purdue.edu/ecetr](http://docs.lib.purdue.edu/ecetr)
PROJECT MODELING FOR DISCRETE EVENT SIMULATION BY NEURAL NETWORKS AND FUZZY MEMORIES

O.E. VELIPASAOGLU
O.K. ERSOY
T. NIENHAUS

TR-EE 92-55
DECEMBER 1992

SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285
Project Modeling for Discrete Event Simulation
by Neural Networks and
Fuzzy Associative Memories

O.E. Velipasaoglu, O.K. Ersoy
and T. Nienhaus

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.1. Durations and total floats of the activities of figure 2.2.1.</td>
<td>77</td>
</tr>
<tr>
<td>5.3.2. Priority values for FAM rules</td>
<td>81</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. Parts of simulation model for project scheduling</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1. Example activity network in A-on-A representation</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2. Event adjacency matrix corresponding to figure 2.1.1</td>
<td>10</td>
</tr>
<tr>
<td>2.1.3. Representations to preserve uniqueness of activity designation</td>
<td>11</td>
</tr>
<tr>
<td>2.1.4. Representations to preserve logical relations</td>
<td>13</td>
</tr>
<tr>
<td>2.1.5. A-on-N representation of the AN of project of figure 2.1.1</td>
<td>14</td>
</tr>
<tr>
<td>2.1.6. Activity adjacency matrix corresponding to figure 2.1.5</td>
<td>15</td>
</tr>
<tr>
<td>2.1.7. Conversion of the event adjacency matrix to the activity adjacency matrix</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1. CPM analysis on the activity network of figure 2.1.1</td>
<td>21</td>
</tr>
<tr>
<td>3.1.1. Generic neuron</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2. Sigmoid $S(x)$, and thresholding nonlinearity $t(x)$</td>
<td>25</td>
</tr>
<tr>
<td>3.1.3. BPNet topology</td>
<td>28</td>
</tr>
<tr>
<td>4.4.1. Columns of $X$ corresponding to the k-th mask</td>
<td>44</td>
</tr>
<tr>
<td>5.1.1. Fuzzy sets and membership functions</td>
<td>69</td>
</tr>
<tr>
<td>5.1.2. FAM system architecture</td>
<td>71</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>5.3.1.</td>
<td>Histograms of activity durations (on top) and total floats (at bottom)</td>
</tr>
<tr>
<td>5.3.2.</td>
<td>Fuzzy sets over ground set &quot;durations&quot; and the membership functions</td>
</tr>
<tr>
<td>5.3.3.</td>
<td>Fuzzy sets over ground set &quot;slacks&quot; and the membership functions</td>
</tr>
<tr>
<td>5.3.4.</td>
<td>Fuzzy sets over ground set &quot;priorities&quot; and the membership functions</td>
</tr>
<tr>
<td>5.3.5.</td>
<td>Operation of BIOFAM</td>
</tr>
</tbody>
</table>
ABSTRACT

The simulation method is used to obtain approximate solutions to operations planning problems. Its power comes from the ability to model complex systems effectively. This property makes simulation preferable in practice over programming methods which provide exact solutions but are limited in the range of problems they can be applied. Flexibility of simulation modeling is desirable especially in resource constrained operations planning problems. When internal decision mechanisms like heuristic methods are added to the scope of simulation, performance can be enhanced significantly.

As a result of application of neural networks and fuzzy associative memories to operations planning through the context of simulation, we expect to improve the plan quality in terms of accurate projection of schedule, cost and performance over the project duration.

For this purpose, we partition the general system model into three parts. We implemented the functions of the activity network, the first part, using a neural network. Our analysis of the proposed
network predicted hundred percent recall success, and we verified this through experiments. The second part which is to resolve resource constraints is implemented with a fuzzy associative memory. This approach allows us to use multiple heuristics to obtain the best results in plan quality. Our work is based on the success of earlier studies of multi-heuristic techniques in solving resource constrained operations planning problems. The rest of the process is taken care of by the third part of simulation model which includes everything that is not included in activity network part or in determining the priorities using heuristics. This part allocates the available resources to pending activities in the order of priorities assigned by the FAM, until the skyline of resources are reached. It puts the activities which did not receive service from a resource to hold status to evaluate its request again when resources become available.
1. INTRODUCTION

Shortly after Hopfield's success with neural networks in 1982, [Hopfie82], the long forgotten area of connectionist intelligence drew upon fresh research interest. Since then, neural networks have been applied to almost all fields where decision making, optimization or modeling is used. Operations planning is one of these fields. Benefits expected include improvement of plan quality in the sense that project progress closely matching the plan in schedule, cost and performance, and retaining corporate memory in manpower.

Exact solutions to operations planning problems are provided by programming approaches. Therefore most of the research effort has been concentrated on translating a programming problem, mainly linear programming, to neural network implementation. One of the first studies is provided by Hopfield and Tank again [TanHop86]. This effort is followed by the works of Zhou et al. [Zhou90], and Yu [Yu90], using the Hopfield network as starting point. Kalaba and Moore took associative memories [KalMoo90] and Barbosa and Carvalho used their own neural network for fast solution of linear systems [BarCar90] as starting points to study the linear programming problem.
Although most of the neural networks studied in these works converge to a solution very quickly, they are susceptible to local minima problems. Hence they don't provide the best solution. In general, real life systems are too complex to model by linear modeling, and inherently so by the neural networks which implement linear programming approach. Some systems cannot be modeled mathematically. And those that can be modeled and analyzed using programming techniques are too simple to be of interest.

Another approach used for operations planning is simulation. Simulation is preferred because it provides the closest model to a real life system. In the context of simulation, one can model systems with resource constraints without difficulty as opposed to programming approaches. When the internal decision mechanisms are missing, simulation gives inexact solutions to operations planning problems. When heuristic methods are introduced for internal decision making, quality of solutions improve. Boctor reports that probability of obtaining best solution can be as high as 95 percent when combinations of heuristics are used [Boctor90].

In order to take advantage of and to possibly improve these properties of simulation, neural networks are applied to simulation modeling. Fishwick categorizes modeling using neural networks under two forms: (a) behavioral and (b) structural [Fishwi89]. Under behavioral form he tries to model a complete system with a single neural network, training the network for regenerating input -
output relation of the original system. He points out that there is no structural relation between the connection scheme of neurons in the network and the system under investigation. Hence he suspects the ability of an algebraic system to realize such a mapping. Under structural form, he translates the structure of a system such as computer network, directly to the structure of neural network. He argues that such a correspondence of structure would be rare for real life systems, and abandons this approach. However this is not necessarily true. In fact, there is a vast amount of real life large scale systems which would lend themselves to such modeling. Most of the real life systems possess an internal structure that can be modeled by using graph theory. Project scheduling problems provide one such group of systems. Elmaghraby analyzes this problem in his state of the art book [Elmagh77]. Sage looks at the problem from general system point of view in his book [Sage78]. Badiru provides a simulation analysis of such a system [Badiru91]. These systems bear the network properties Fishwick considers necessary to be able to create a neural network model of the system. Schmidt, Haddock and Wallace compare three approaches for modeling: (a) discrete event simulation (statistical modeling), (b) object-oriented simulation, and (c) neural network simulation [ScHaWa91]. They report that behavioral modeling works well when the neural network is trained using good case studies.

Both simulation and the programming approach implemented with neural networks provide inexact solutions to the operations planning problem. On the other hand, simulation can model complex
systems more closely than programming can do. Our literature survey shows that application of neural networks to simulation modeling has not been studied in depth and have been overlooked at some promising details. Therefore, we concentrate our research effort in this direction. We will look at the project scheduling problems since they provide a large group of examples of real life systems that can be analyzed by simulation.

Both of the studies mentioned above using behavioral form try to model a complete system by a single neural network. The system is complicated by the presence of resource constraints. Elmaghraby starts analyzing such systems by disregarding the effects of resources. He comes up with a graph model of the system called activity network. After this simplification, he introduces the resource constraints back. Such a simplification may greatly enhance the performance of a neural network model. Therefore we partition the system under investigation down to simpler sub-systems and model them separately, instead of modeling the complete system at once. First, partitioning should separate the activity network and the resource constraints. We also create a third part which includes everything not included in activity network part or the resource constraints part. For a typical system, this partition would look like as in figure 1.1. Also, the relations between parts are given in this figure. We treat each part as an object, which effectively combines the approach in [ScHaWa91]. This feature will introduce the plusses of object-oriented simulation to our approach. Being implementable on a parallel distributed system would be one such advantage. In
fact we can further divide the parts into sub-parts and treat them as objects to increase granularity. Distributed simulation of resource constrained project scheduling is studied in a similar way by Arora and Sachdeva [ArSa89].

![Diagram of simulation model for project scheduling.](image)

**Figure 1.1.** Parts of simulation model for project scheduling.

We propose a neural network implementation for the activity network part. As we mentioned before, integrating heuristic methods in simulation for solving resource constrained problems results in good performance. Fuzzy Associative Memories (FAMs) are very suitable for implementing linguistic expert rules. Therefore we propose a FAM implementation for the resource constraints part.
FAMs are implementable using neural network architectures. Therefore they are considered as neural networks, too.

In Chapter 2, we outline the properties of systems which can be partitioned into activity network and resource constraints parts. In the first section of Chapter 2, we investigate the activity networks and in the second section, we introduce the properties of resource constrained planning. Chapter 3 summarizes definitions for neural networks and briefly discusses the Delta Rule Memory which will be used in this thesis. In Chapter 4, we introduce the neural network solution for the activity network modeling problem. In Chapter 5, we outline the theory of fuzzy sets and discuss FAMs briefly. In Chapter 6 we give the FAM solution to resource constraints modeling. Chapter 7 is discussions and conclusions.
2. PARTS OF PROJECT MODEL

2.1 Activity Network

For analysis of a real world project scheduling problem by simulation a model describing the project. A project is modeled as a collection of activities and events. An activity is any undertaking that consumes time and resources. Examples of activities are writing a thesis, doing a homework, building the foundation of a house, etc. An event is a well-defined occurrence in time. Examples of events are taking the final examination, submitting the homework, receiving the shipment, etc.

There exists a precedence relation between activities, stemming from causality principles that govern real world phenomena. Every activity has a duration associated with it. Duration might be considered as deterministic or probabilistic. In the context of simulation, it is often considered probabilistic. Each activity consumes certain resources. Types of resources are (a) financial resources, (b) labor, engineering and managerial skills, (c) machinery and equipment, and (d) natural resources like energy, land, water, materials, etc. Effects of consuming different types of resources can be summed in a single cost variable which is associated with each activity.
The precedence relationship between activities possesses certain properties:

1. Precedence is basically a binary relationship: An activity \( p \) precedes another activity \( q \), which is written as \( p \succ q \), meaning that \( p \) must be completed before \( q \) is started; or a set of activities \( P \) precedes another set of activities \( Q \), also written \( P \succ Q \), in which case the internal order of activities in either set is immaterial, but no activity in \( Q \) may be started before all the activities in \( P \) have been completed. We may transform this binary relation between sets into a binary relation between elements of sets in the following manner. Let \( P_l \) denote the activity accomplished last in the set \( P \), and let \( Q_f \) denote the activity ranked first among the activities in the set \( Q \). Then the condition \( P \succ Q \) is equivalent to \( P_l \succ Q_f \).

2. Precedence is a transitive relation; in other words, if \( p \succ q \) and \( q \succ r \), then \( p \succ r \).

When graphs are used for representing the activities, events and relationships between these elements, the model becomes an activity network (AN). There are two possible modes of representation for ANs: (a) activity-on-arc (A-on-A) representation and (b) activity-on-node (A-on-N) representation. Both representations are directed graphs (or digraphs).

(a) The Activity-on-Arc Representation:

An activity is represented by an arc, and an event by a node. An example of activity network is given in figure 2.1.1.
Such an AN has the following characteristics:

1- The nodes are numbered such that an arrow always leads from a small number to a larger one. This is possible when the digraph is acyclic. An immediate consequence of such a numbering scheme is that the adjacency matrix representation of the network is always upper triangular with zero diagonal. Adjacency matrix is an nxn matrix, where n is the number of nodes in the network and has an entry "1" for element \((i, j)\) if an arrow leads from node \(i\) to node \(j\), and "0" otherwise. The adjacency matrix is thus a Boolean matrix. It is convenient to call this adjacency matrix an event.
adjacency matrix since it bears the information about adjacent nodes which represent events in A-on-A representation. Event adjacency matrix of the network of figure 2.1.1 is shown below.

$$
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 2.1.2. Event adjacency matrix corresponding to figure 2.1.1.

2- Each node (event) must have at least one arc leading into it and one arc going out of it, with the exception of the origin (node 1, or source) and the terminal (node n, or sink). The former has only arcs leading out of it and the latter arcs leading into it.

Any two nodes may be connected by at most one arc. This constraint can be satisfied in one of two ways: (a) if several activities must take place simultaneously (i.e., in parallel) between two nodes, they can be combined in one global activity or (b) use dummy activities (of zero duration and zero consumption of resources) and dummy events as shown in figure 2.1.3.
3- By construction, the network contains no cycles, since otherwise by tracing around the cycle we would conclude, by transitivity property of precedence, that a task must precede itself, which is a contradiction. Therefore, in any forward path from origin to terminal, each node on the path appears once and only once.

Another constraint imposed is that each network must possess one and only one origin and one and only one terminal. This condition can always be satisfied by adding one dummy event, node 1, which connects by dummy activities to all nodes of the network with no
predecessors, and a dummy event, node n, which connects by dummy activities from all nodes of the network with no successors.

As a consequence of such a construction, each node (event) of the network is connected from the origin by at least one path and is connected to the terminal by at least one path.

4- Since the precedence relationship is transitive, each node (event) along a path must have all preceding activities completed (and all preceding events realized) before it can be realized. In fact this condition is the definition of "the realization of a node".

Because of strict logical interpretation of the arrows, great care must be taken in drawing the network lest a precedence relationship is either added to or omitted from, the original specifications of the project. For instance, consider four activities a, b, c, and d related by precedence as

\[ a > c, \ b > c, \ b > d \]

At first try, one might construct the network in figure 2.1.4, on top. However close scrutiny reveals that such a representation also implies that \( a > d \), which was not specified. the correct representation is shown in figure 2.1.4, on bottom, in which x is a dummy activity of zero duration.
Figure. 2.1.4. Representations to preserve logical relations.

(b) The Activity-on-Node Representation (A-on-N):

An activity is represented by a node and the precedence relationship between two activities is represented by a directed arc in which the direction of the arrow specifies the precedence. In this mode of representation one may consider an "event" to be represented by the arc since, in some sense, "movement" from one node to another can take place only after the completion of the former task. Such a state completion defines an event.
The (A-on-N) representation of the project of figure 2.1.2 is shown in figure 5. The corresponding adjacency matrix is given below. If the project has A activities, the adjacency matrix is an \( A \times A \) matrix with entry "1" if the activity \( u \) precedes activity \( v \), and "0" otherwise. \textbf{Therefore} it is convenient to call this matrix the activity adjacency matrix. This matrix is also upper triangular with proper numbering of activities.

![Diagram](image-url)

Figure. 2.1.5. A-on-N representation of the AN of project of figure 2.1.1.
A-on-A and \textbf{A-on-N} representations are dual of each other and transferring from one representation to the other is straightforward.

In this work, we will concentrate on the graph representation of the A-on-A mode. Both event and activity adjacency matrices will be used. Converting from upper triangular event adjacency matrix to upper triangular activity adjacency matrix is described below.
There is a unique numbering of activities which will produce the upper triangular activity adjacency matrix given the event numbering in A-on-A representation, and hence the event adjacency matrix.

Event adjacency matrix is traversed reading each row from left to right, from first row to last. A counter is initialized to zero before starting the process and incremented each time a "1" is encountered. After incrementing the counter the entry "1" is replaced with the value of the counter. This process gives the unique numbering of the activities.

Then the precedence relations are read from the updated event adjacency matrix. Successor activities of each event node are the non zero entries on the row corresponding to the event number. Predecessor activities of each node are the non zero entries on the column corresponding to the event number.

This procedure is demonstrated with the event adjacency matrix of the project of figure 2.1.2 in figure 2.1.7.
Event adjacency matrix corresponding to figure 2.1.1.

\[
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Updated event adjacency list. (Zeros are not shown.)

Figure 2.1.7 Conversion of the event adjacency matrix to the activity adjacency matrix.
Predecessor activities for each event. (Zeros not shown.)

Successor activities for each event. (Zeros not shown.)

Figure 2.1.7 Continued.
2.2. Resource Constraints

When the resource constraints are not present, in other words unlimited availability of each resource is assumed, the operations plan for minimum project duration can be determined using Critical Path Method (CPM) or Project Evaluation and Review Technique
CPM assumes that the activity durations are deterministic. It obtains the operations plan by a forward pass followed with a backward pass through the digraph of the activity network. To demonstrate this, we will assign durations to the activity network of figure 2.1.1 (figure 2.2.1). During the forward pass earliest possible occurrence times of the events are calculated. This number is the maximum path length from the origin to that node, where path length is the sum of the durations of the activities on the forward path from the origin to that node. Backward pass is to calculate the latest possible occurrence times of the events. This number is equal to the maximum of backward path length from the sink node to that node subtracted from the earliest occurrence time of the last event. Critical path is the group of activities whose start and end nodes have the same earliest and latest possible occurrence times. Using the possible occurrence times of the event nodes, a bunch of float (also called slack) definitions can be made. Float is the extra time available for an activity. Most important floats are free float and total float. Free float is the duration of the activity subtracted from the difference of earliest occurrence times of start and finish nodes. Total float is duration activity subtracted from the maximum time available for it, which is the difference between earliest occurrence time of the start node and latest occurrence time of the finish node. Notice that for the activities on the critical path, total float is zero.
When the resource constraints are introduced back to the problem, solutions provided by these methods will be infeasible in most of the cases. Some of the activities will not be scheduled at the times predicted by the operations plan provided by these methods simply because the necessary resources are allocated to activities which have started earlier and have not finished yet. Therefore start times of these activates will be delayed until resources are available. Project time will increase if these activities are on the critical path.
Furthermore, the critical path may change leaving the former analysis obsolete. Problem stems from the fact that CPM or PERT does not care about the resource skylines while optimizing the operations plan for the project duration. On the other hand, even if the operations plan remains feasible after the resource constraints are introduced back, it will probably not be an optimum plan anymore. An operation plan which includes resource utilization and peak/total resource requirements into account will be a better one. Minimizing the idle time of resources, cutting the peak resource requirements to meet with the total resource requirements at a level are desirable. Kurtulus and Davis define two performance measures for these objectives [KurDav82].

As we mentioned in the introduction, the best solution to the resource constrained operations planning problem in terms of time complexity and solution quality is provided by simulation methods combined with heuristic methods. If the heuristics were not present, a resource allocation tie would be broken by randomly assigning the available resource to one of the tasks. However if for example one of the tasks is on the critical path, allocating the resource for that task makes more sense at first look. The reason for this is if the resource is allocated to the task which is not on the critical path, the task on the critical path would be delayed and the project time would increase directly. On the other hand, since the task which is not on the critical path has certain amount of slack, it could tolerate the delay. This is the flavor behind the heuristics as is behind all the
greedy methods. Relation to critical path could be one useful heuristic rule. Some of the other heuristic rules include:

MINSLK: Activity having the smallest slack is scheduled first.
RSM: Resource scheduling method [BrMeSh64].
MINLFT: Give precedence to the activity having the smallest finish time (as given by the critical path analysis).
RAN: Select an activity randomly.
GRD: Give priority to the activity which requires the greatest number of resource units of all types.
SA: Choose first the shortest activity.
SRD: Smallest resource demand (the inverse of GRD).
LA: Choose first the longest activity.

We will use the same idea, namely we will assign priorities to the activities demanding resources using the heuristics defined above.

Boctor shows that using more than one heuristic at the same time improves the resulting operations plan and investigates certain combinations of heuristics [Boctor90]. Since we will implement the heuristics using FAM we will be using more than one heuristic at the same time automatically.
3. NEURAL NETWORKS AND DELTA RULE MEMORY

This chapter will briefly describe the mathematical concepts used to analyze neural network (NNet) processing in the first section and Delta Rule Memory in more detail in the second section.

3.1. Neural Networks

In a broad sense, a NNet consists of three elements (1) an organized topology of interconnected neurons (processing elements), (2) a method of encoding information (learning), and (3) a method of recalling information (recalling).

Neurons, also referred to as nodes, are units where most of the computing is done. Figure 3.1.1 displays the anatomy of a generic neuron. The input signals come either from environment as inputs to the NNet or from other neurons. Sum of the weighted input signals is called the activation of the neuron. Output of the neuron is the activation value passed from a nonlinearity. Common nonlinearities are thresholding function and sigmoid function (figure 3.1.2).
Figure 3.1.1. Generic neuron.

\[ t(x) = \text{signum}(x) \]

Figure 3.1.2. Sigmoid \( S(x) \), and thresholding nonlinearity \( t(x) \).
There are two connection types: excitatory and inhibitory. Excitatory connections increase neurons activation and are typically represented by positive connection weights. Inhibitory connections decrease neurons activation and are typically represented by negative connection weights. A group of neurons is called a layer. The three primary interconnection schemes are intra-layer, inter-layer and recurrent connections. Intra-layer connections connect neurons in the same layer. Inter-layer connections connect neurons in different layers. Recurrent connections are connections that loop and connect back to the same neuron. Inter-layer connection signals propagate in two ways: (1) Feedforward signals only allow information to flow amongst neurons in one direction, (2) Feedback signals allow information flow in either direction and/or recursively.

A field that receives input signals from the environment is called an input layer, and a field that emits signals to environment is called an output layer. Any fields that lie in between are called hidden layers.

A NNet stores spatial or spatiotemporal patterns. An associative memory maps data to data. A NNet network can be used as either an autoassociative or a heteroassociative memory. Autoassociative memory can be viewed as a special case of a heteroassociative memory.

All learning methods can be classified into two categories, supervised learning and unsupervised learning. Supervised learning is a process that incorporates an external teacher and/or global
information. Unsupervised learning, also referred to as self-organization, is a process that incorporates no external teacher and relies upon only local information and internal control.

For our application we will briefly discuss multilayer feedforward neural networks with supervised learning in some more detail below. We will discuss Backpropagation network (BPNet) since it is the most popular paradigm.

The elementary BPNet is a three-layer NNet with feedforward connections from the input layer to hidden layer and feedforward connections from the hidden layer to output layer. In general, it is possible to have several hidden layers, and more complex connection schemes. The elementary BPNet is a heteroassociative, function-estimating NNet that stores arbitrary analog spatial pattern pairs \((A_{k}, C_{k})\), \(k=1,2,\ldots,m\), using a multilayer gradient descent error-correction encoding algorithm (also called the Backpropagation learning algorithm (BP), or the generalized delta rule), where the \(k\)-th pattern pair is represented by the vectors \(A_{k}=[a_{1}, \ldots, a_{n}]\) and \(C_{k}=[c_{1k}, \ldots, c_{qk}]\). BPNet learns off-line using a supervised learning algorithm, operates in discrete time, and is represented in figure 3.1.3 by the three-layer feedforward topology, where the \(n\) neurons in the input layer correspond to the components of \(A_{k}\) and the \(q\) neurons in the output layer correspond to the components of \(C_{k}\).
Recall is in the following way: Let $W_1$ be the matrix connection of connection weights from the input layer to hidden layer such that $W_1(i, j)$ is the strength of the connection from i-th input-layer neuron to j-th hidden-layer neuron. Similarly let $W_2$ be the matrix of connection weights from the hidden layer to the output layer. Let
\(X_0 = [x_{01}, \ldots, x_{0n}]\) be the vector representing the values of the input-layer neurons. Similarly let \(X_1 = [x_{11}, \ldots, x_{1t}]\) and \(X_2 = [x_{21}, \ldots, x_{2q}]\) represent the values of the hidden-layer neurons and output layer-neurons respectively. Let \(f_1(.)\) be a vector function such that i-th component \(x_i\) of the argument vector is mapped to the i-th component \(y_i\) of the result vector via function \(f_{ii}(x_i) = y_i\) where \(f_{ii}(.)\) is the nonlinearity of the i-th neuron in the hidden layer. Similarly let \(f_2(.)\) be the vector function bearing the nonlinear transformation for the output layer. Than for an input \(X = [x_{11}, \ldots, x_{nt}]\) presented to the input layer the result \(Z = [z_{11}, \ldots, z_{qk}]\) at the output layer is obtained as follows:

\[
Z = f_2(W_2 \times Y),
\]

where \(Y\) is given by

\[
Y = f_1(W_1 \times X).
\]

BP is not guaranteed to find the global minimum during training but only the local error minimum. This situation often leads to oscillations in the weight changes during training. The key to solve this problem is multifaceted. Several key questions must be answered concerning the proper selection of the number of hidden layer neurons, the size of the learning rate parameters, amount of data necessary to create the proper mapping and the initial values of the connection weights.

Several studies have independently found that a three-layer \textbf{NNNet} using the BP algorithm can approximate a wide range of
functions to any desired degree of accuracy [IrieMi88]; [MooPog88].

BP's strengths include its ability to store many more than the number of input layer dimensions \((m >> n)\) and its ability to acquire arbitrarily complex nonlinear mappings. Although the deduction mechanism of the N Nets is not known explicitly, BPNet is capable of producing useful generalizations. Generalization occurs when the NNet is presented an input it has not seen during the training phase. The output is one of the stable states created during the training. If the output is meaningful this is called a useful generalization. If the output is not meaningful or it is not close to the desired output in the case of supervised training, the output is called a spurious state.

3.2. Delta Rule Memory

When hidden layer is not present the BPNet reduces to a Delta Rule Memory (DRM). The learning algorithm is the delta rule instead of generalized delta rule. In the following analysis we will follow a close path to that in Ersoy's work [Ersoy88], [BusErs90].

Let \(X\) be a collection of \(K\) \(M\)-bit input vectors. Similarly let \(D\) and \(Y\) be collections of \(K\) \(N\)-bit desired output and actual output vectors, respectively.

\[
X = [x^1, \ldots, x^k, \ldots, x^K] \\
Y = [y^1, \ldots, y^k, \ldots, y^K] \\
\text{where} \quad x^k = \begin{bmatrix} x^k_1 \\ \vdots \\ x^k_M \end{bmatrix}
\]
\[ Y = \left[ y^1, ..., y^k, ..., y^K \right] \quad \text{where} \quad y^k = \begin{bmatrix} y^k_1 \\ \vdots \\ y^k_N \end{bmatrix} \] and

\[ D = \left[ d^1, ..., d^k, ..., d^K \right] \quad \text{where} \quad d^k = \begin{bmatrix} d^k_1 \\ \vdots \\ d^k_N \end{bmatrix}. \]

Since there is no hidden layer the output is expressed as:

\[ Y = f(W \times X) \]

\( W \) is the connection weights matrix. \( w(i,j) \) is the connection strength from the \( j \)-th input node to \( i \)-th output node, and \( f(.) \) is the output nonlinearity.

Delta rule learning algorithm minimizes the following error function:

\[
E(W) = \sum_{k=1}^{K} \left| d^k - y^k \right|^2
= \sum_{k=1}^{K} \sum_{i=1}^{N} (d^k_i - y^k_i)^2
= \sum_{k=1}^{K} \sum_{i=1}^{N} (d^k_i - f(a^k_i))^2
\]

where \( a^k_i \) is the activation of the \( i \)-th output neuron for the \( k \)-th input vector, defined as:

\[ a^k_i = \sum_{j=1}^{M} w_{ij} x^k_j \]
To minimize this function we must set its derivative equal to zero:

\[
\frac{\partial E(W)}{\partial w_{ij}} = \sum_{k=1}^{K} \sum_{i=1}^{N} \frac{\partial}{\partial w_{ij}} \left( (d_i^k - f(a_i^k))^2 \right)
= (-2) \sum_{k=1}^{K} \sum_{i=1}^{N} (d_i^k - f(a_i^k)) \frac{\partial}{\partial w_{ij}} (f(a_i^k))
\]

In order to be able to continue with this analysis we have to assume that the nonlinearity is differentiable. Therefore we prefer sigmoid over threshold:

\[
f(a_i^k) = \frac{1}{1 + e^{-s}}
\]

with s as the parameter controlling the steepness of the sigmoid.

Then the derivative of the error expression becomes:

\[
\frac{\partial E(W)}{\partial w_{ij}} = (-2) \sum_{k=1}^{K} \sum_{i=1}^{N} (d_i^k - f(a_i^k)) \frac{se^{-s}}{(1 + e^{-s})^2} x_i^k
\]

Another way to view this problem is to map the desired outputs to desired activations. Let \( e_i^k \) be the desired activation corresponding to the desired output \( d_i^k \). Assuming that the output nonlinearity is invertible:

\[
f(e_i^k) = d_i^k
\]
\[
e_i^k = f^{-1}(d_i^k)
\]

Output nonlinearities of neural networks are monotone nondecreasing. Therefore differentiability assumption and the invertibility assumption made above are equivalent. Again we
prefer sigmoid over threshold. There is one more point to be taken care of: Inverse of sigmoid grows unbounded. However for practical reasons we can assume that the range space is smaller than \((0,1)\) interval. Reaching the range boundaries at the output requires unbounded activation, hence this assumption is not bad. We will confine the range into the \([0.1,0.9]\) interval.

Now we can express the error function once more:

\[
E(W) = \sum_{k=1}^{K} \left[ f^{-1}(d^k) - f^{-1}(y^k) \right]^2
\]

By differentiating both sides we obtain:

\[
\frac{\partial E(W)}{\partial w_{ij}} = -2 \sum_{k=1}^{K} \sum_{i=1}^{N} (e_i^k - a_i^k)x_j^k
\]

Since \(f\) is a monotone nondecreasing function both of the gradients obtained above points to the same direction. Therefore we can use either of them for the definition of the steepest descent method:

\[
Aw, = -\alpha \frac{\partial E(W)}{\partial w_{ij}}
\]

where \(\alpha\) is a scaling constant.
To compute the gradient all the input vectors are needed because of the summations. In practice approximation to the true gradient is used:

\[ \Delta w_{ij} = \alpha_k (d^k_i - y^k_i)x^k_j \quad \text{or} \quad \Delta w_{ij} = \alpha_k (e^k_i - a^k_i)x^k_j \ . \]

Expressed in the vector form:

\[ \Delta W = \alpha_k (d_i - y_i)(x^i)^t \quad \text{or} \quad \Delta W = \alpha_k (e_i - a_i)(x^i)^t \ . \]

When \( \alpha_k \) is small and a number of passes over the whole data set are made using these expressions, a good approximation to the true gradient is obtained.

Again, \((d^k_i - y^k_i)\) and \((e_i - a_i)\) have the same direction, hence the approximation to the gradient differs by a scalar between two options. This difference can be absorbed into the constant \( \alpha_k \) which scales the norm of the approximation. Therefore two ways of obtaining the weight matrix should give the same results.

It is not apparent from the first expression for the error function but if we look at the second form of the same error function we recognize the familiar form of the least squares optimization:

\[ E(W) = \sum_{i=1}^{K} |e_i - a_i|^2 \ . \]

The goal is to satisfy:

\[ a^k = Wx^k = c^k = f^{-1}(d^k) \ . \]
Using the matrix notation:

\[ WX = E \]

where \( E = [e^1 \ldots e^k \ldots e^K] \) and \( e^k = [e^1_k \ldots e^N_k] \).

The solution is

\[ W = EX^* \]

\( X^* \) is the pseudoinverse of \( X \).

When a test vector \( p \) is presented to the system, it is first multiplied by \( X^* \). This results in projection of \( p \) into column space of \( X \):

\[ q = Wz \]
\[ = EX^*p \]
\[ = EX^*(p_{||} + p_{\perp}) \]
\[ = Ep_{||} \]

\( p_{||} \) is the projection of \( p \) and \( p_{\perp} = p - p_{||} \). When the projection is not unique, the pseudoinverse picks the optimal solution which has the minimum length. The activation vector is a linear combination of the columns of \( E \) weighted with elements of the projection. The actual output is the activation vector transformed by the output nonlinearity. When the nonlinearity is sigmoid which is monotone increasing and one-to-one, the output is the nonlinearly amplified version of the activation vector.
4. NEURAL NETWORK SOLUTION FOR ACTIVITY NETWORK: MODELING

We start this chapter by describing the problem. Then we introduce our solution and present the experiment results.

4.1. Description of the Problem and Some Notation

In DES, simulation clock runs in virtual time, and is updated only when an event occurs. An event is said to have occurred when all the activities leading to it are finished. Therefore one of the main concerns is to be able to tell from the model, if an event has occurred when a set of finished activities is presented. Also, in order to keep the simulation clock running, successor activities of the events which have occurred must be scheduled. If we rephrase the problem, we have to be able to determine from the model, the set of activities to be scheduled when a set of finished activities is presented.

Next we introduce some notation for the analysis of the problem:

\[ a := \text{an activity in the project} \]
\[ e := \text{an event in the project} \]
\[ n, := \text{total number of activities in the project} \]
\[ n_e := \text{total number of events in the project} \]
$S_i := \{ a | "a" \text{ is an activity in the project} \}$

$\xi_k := \{ a | "a" \text{ is a fan-in activity for event } k \}$ \hspace{1cm} k=1,\ldots,n_e

$\varphi_k := \{ a | "a" \text{ is a fan-out activity from event } k \}$ \hspace{1cm} k=1,\ldots,n_e

$i_k := \text{ number of fan-in activities to event } k$

$o_k := \text{ number of fan-out activities from event } k.$

Properties of $S_i,$ $\xi_k,$ and $\varphi_k :$

i- $\bigcup_{k=1}^{n_e} \xi_k = S_r - \{\text{sink}\}$ \hspace{1cm} where sink is the last activity in the project.

ii- $\xi_i \cap \xi_j = \emptyset$ if $i \neq j$

iii- $\bigcup_{k=1}^{n_e} \varphi_k = S_r - \{\text{source}\}$ \hspace{1cm} where source is the first activity

iv- $\varphi_i \cap \varphi_j = \emptyset$ if $i \neq j$

Let $id(.)$ be function such that it takes an activity as argument and maps it to an integer. The mapping is as described in the second chapter. Let $b(.),$ $g(.)$ be functions which map an unsigned integer to its binary representation, and Gray coding scheme representation respectively. Similarly, let $s(.)$ be a function which maps an unsigned integer to its simple coding scheme representation. If there are $n$ integers in the domain to be encoded, simple scheme uses $n$ bits for representation. If "i" is an unsigned integer, representation of "i" in simple scheme is a $n$-bit vector with all the elements zero except for the i-th element which is "1".
4.2. **Representation Scheme**

For our problem, we need a representation of sets of activities which will put the input in a format presentable to NNets. Most popular schemes are binary and gray coding ([Lawren91], [TakGoo86]). These schemes are preferred because they encode information in relatively small vectors, resulting in smaller networks. Both schemes encode \( n \) items in \( \lceil \log_2(n) \rceil \) bits, where \( \lceil u \rceil \) is the least integer larger than \( u \). We can define a representation function, \( \text{rep}(.) \), using either one of these schemes, however domain of \( \text{rep}(.) \) should be the power set of \( S \), which has \( 2^n \) elements. Therefore we need \( \log_2(2^n) = n \) bits for representation. Therefore binary (or similarly Gray) encoding does not provide more compact representation than the simple scheme. Furthermore when binary or gray encoding is used back and forth conversion is not simple for the elements of the power set of \( S \), other than those with single element.

\[\begin{align*}
\text{e.g. } & \text{ Assume } n \text{ is 3.} \\
\text{rep}(\{1\}) = & \text{ b(1) = 01} \\
\text{rep}(\{2\}) = & \text{ b(2) = 10} \\
\text{rep}(\{3\}) = & \text{ b(3) = 11} \\
\text{rep}(\{1,2\}) : & \text{ cannot be defined straightforwardly unless 4 bits are used.}
\end{align*}\]
Simple scheme solves this problem naturally. We simply let

$$\text{rep}([a,b]) = \text{rep}([a]) + \text{rep}([b]) = s[id(a)] + s[id(b)]$$

This definition is valid because $s[id(a)]$ and $s[id(b)]$ will have "1"s in different locations, so the sum will be new vector which can be allocated for $\text{rep}([a,b])$. Therefore we define our representation function as :

$$\text{rep}(F) = \sum_{a \in F} \text{rep}([a]) = \sum_{a \in F} s[id(a)]$$

In fact this is the most compact and straightforward representation scheme we can use for the power set of $S$.

Last definition is the definition for a mask. Let $m_k$ be a $n_k$-bit mask vector for the event $k$, $k=1,\ldots,n$. We define $m_k$ as :

$$m_k = \text{rep}([\xi_k])$$

4.3. **Training Set**

Now that we have chosen the encoding scheme, we can proceed to obtain the solution to the problem. Since we are going to use DRM with a supervised learning rule, we will need data pairs for training. Let $X=[x^1,\ldots,x^i,\ldots,x^T]$ be a matrix where $x^i$ is a column vector corresponding to antecedent vector of the $i$-th data pair. Similarly, let $E=[e^1,\ldots,e^i,\ldots,e^T]$ be a matrix where $e^i$ is the column vector for desired activations. $T$ is the total number of data pairs in the training set. Once we determine $X$ and $E$, the weight matrix will be
\[ W = EX^* \]

as we obtained in the third chapter.

A candidate for the training set is

\[ \Psi = \bigcup_{k=1}^{n} \hat{P}(\xi_k) \]

\( \hat{P}(A) \) is the power set \( P(A) \) of set \( A \) minus the empty set:

\[ \hat{P}(A) = P(A) - \emptyset \]

Hence

\[ x^i = \tau_{\text{rep}}(P(i)) \quad i=1, \ldots, T \]

In this equation it is assumed that \( P \) is an ordered set, so \( P(i) \) is the \( i \)-th element of \( P \). There is no loss of generality in this assumption. Then

\[ T = \sum_{k=1}^{n} (2^k - 1) \]

Desired output set is defined as follows:

\[ d^i = \begin{cases} \tau_{\text{rep}}(\varphi_k) & \text{if } 3k \in \{1, \ldots, n_e\} \text{ such that } P(k) = \xi_k \\ 0 & \text{otherwise} \end{cases} \]

Elements of the desired activations \( e^i \) are elements of these vectors mapped back through inverse of sigmoid function.

Observe that \( P \subset \hat{P}(S_e) \) and \( P \not\subset \hat{P}(S_e) \). The reason for the latter is that in a valid activity network at least one of the activities
must be a fan-out activity which would be the sink. This observation means that we expect the network to be able to generalize correctly for the elements in the difference set \( \hat{P}(S_i) - P \neq \emptyset \).

The reason for choosing \( P \) as such is explained as follows. Notice that the information for deciding if an event has occurred must be present in the fan-in set corresponding to that event. Namely, an event is said to have occurred if all the activities, but not less, in the fan-in set have been finished. Similarly, the information for deciding which activities to schedule after an event has occurred must be present in the fan-out set corresponding to that event. The activities to be scheduled are the elements of the fan-out set.

We restrict our domain to the inputs from the set \( P \) for improving our analysis. We will later return to the analysis of generalization for the inputs in the difference set \( \hat{P}(S_i) - P \).

Since the input is an element of \( P \), using properties i - iv of the sets \( S_i, \xi_i, \) and \( \varphi_k \), we can confine our analysis to the set \( \hat{P}(\xi_k) \) for an arbitrary \( k \) without loss of generality. In other words we will consider the vectors which are embedded in an arbitrary single mask.

There are \( 2 - 1 \) columns in \( X \) corresponding to the elements of \( \hat{P}(\xi_k) \). Only one of the columns in \( D \) corresponding to these columns is nonzero: The column in \( X \) corresponding to \( \xi_k \), which is \( m, = \text{rep}\{\xi_k\} \), is associated with \( \text{rep}(\varphi_k) \) in \( D \). The rest of the columns in \( X \) are associated with \( 0 \) in \( D \).
To be able to express the elements of the power set of $\xi_k$, we need the following definition:

$$\rho^n(\xi_k) = \{ F \mid F \subseteq \xi_k \text{ and } F \text{ has } n \text{ elements} \}$$

Properties of this definition are:

i- $\rho^1(\xi_k) = \xi_k$

ii- $\bigcup_{m=1}^{n} \rho^m(\xi_k) = \bar{F}(\xi_k)$

iii- $\rho^m(\xi_k) \cap \rho^n(\xi_k) = \emptyset$ if $i \neq j$

iv- for $n \geq m$, $\forall \sigma_i \in \rho^n(\xi_k)$ $\exists \sigma_j \in \rho^m(\xi_k)$ such that $\sigma_i \subseteq \sigma_j$

There seems to be a boundary between the elements of $\rho^i$ and $\rho^{i-1}$. Every element of $\rho^i$, with $i \leq i - 1$, is a subset of one of the elements of $\rho^{i-1}$ (from iv), hence we would expect them to lie on the same side of the boundary. This is in fact the case here since any element in $\rho^i$ for $i = 1, \ldots, i_k - 1$, is associated with 0 vector. This observation suggests that we can shrink the size of the training set further.

We propose that shrinking the training set to include only the elements of $\rho^i$ and $\rho^{i-1}$ will be enough for successful recall.
4.4. Analysis of Recall Performance

In order to prove our assertion, we have to change notation from that of sets to vectors. Let every element of $\rho^k$, $i = 1, \ldots, i_k$, be associated with a vector in the following way:

$$x^k_i = \text{rep}[p^i(j)] \quad i = 1, \ldots, i_k; \quad j = 1, \ldots, \begin{pmatrix} i_k \\ i \end{pmatrix}$$

Again we assume that $p^i$ is an ordered set, without loss of generality. $\begin{pmatrix} i_k \\ i \end{pmatrix}$ is the i-combination of $i_k$.

Observe that $x^k_{i_k} = m_i$. Then the training set matrix $X$ will contain

$$x^k_{i_{k-1}}, x^k_{(i_k-1)_k}, \ldots, x^k_{(i_k-1)_{i_{k-1}}}$$

among its columns for the k-th mask. To simplify the notation we will use the set $x_0, x_1, \ldots, x_n$ where

$$n = \begin{pmatrix} i_k \\ i_k - 1 \end{pmatrix} = i_k$$

and

$$x_0 = x^k_{i_k}$$

$$x_i = x^k_{(i_k-1)_k}, \quad i = 1, \ldots, \begin{pmatrix} i_k \\ i_k - 1 \end{pmatrix}$$

The columns of $X$ corresponding to the k-th mask are shown in figure 4.4.1. These columns in $X$ span the n dimensional space. There are $n+1$ columns, so they are linearly dependent. As we explained in the third chapter, first part of $W$ multiplying a test vector $t$ is the
pseudoinverse of $X$. This multiplication will try to express the test vector in terms of columns of $X$.

\[
W_t = EX^*t
\]

\[
X^*t = X^*(t_{11} + t_{12}) = X^*t_{11}
\]

\[
t_{11} = \sum_{k=0}^{n} u_k x_k^* \quad \text{for some } k.
\]

$t_{11}$ is the error associated with projecting $t$ into the column space of $X$. Since columns of $X$ span the $n$ dimensional space, there is no projection error. However, since they are linearly dependent, the expression for the projection will not be unique. The pseudoinverse chooses the projection with the smallest length.

For the following analysis we can neglect the blocks of zeros above and below the $n$ by $n+1$ dimensional submatrix in $X$. Also we
will have to modify our representation for the desired output vectors. In the analysis of learning rule for DRM, in the third chapter, we mapped the desired output values back through the inverse of the sigmoid function to obtain the desired activation values. The desired activation value is

$$\text{sig}^{-1}(0.9) = \frac{1}{10} \ln\left(\frac{1}{0.9} - 1\right) = 0.22$$

for the desired output value of 0.9. Similarly, the desired activation value for desired output value 0.1 is

$$\text{sig}^{-1}(0.1) = \frac{1}{10} \ln\left(\frac{1}{0.1} - 1\right) = -0.22.$$ 

Therefore, the target values to be generated after multiplication of a test vector with the weight matrix are $\pm 0.22$.

Now we will calculate the activation values for a general test vector using row operations. Notice that in the matrix below $x_0$ is placed to the last column to ease the column operations.
\[
\begin{bmatrix}
1 & 1 & 1 & & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & & 0 & 0 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & & 0 & -1 & 0 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & 0 & -1 & & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
t_1 \\
t_2 - t_1 \\
t_3 - t_1 \\
t_4 - t_1 \\
\vdots \\
t_{n-2} - t_1 \\
t_{n-1} - t_1 \\
t_n \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 1 & & 0 & 0 & 0 & 0 & -1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & & 1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 1 & 1 & & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
t_1 - t_n \\
t_1 - t_{n-1} \\
t_1 - t_{n-2} \\
\vdots \\
t_1 - t_5 \\
t_1 - t_4 \\
t_1 - t_3 \\
t_1 - t_2 \\
t_n \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 1 & & 0 & 0 & 0 & 0 & -1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & & 1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & & 0 & 0 & 0 & 0 & n-1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
t_1 - t_n \\
t_1 - t_{n-1} \\
t_1 - t_{n-2} \\
\vdots \\
t_1 - t_5 \\
t_1 - t_4 \\
t_1 - t_3 \\
t_1 - t_2 \\
\gamma \\
\end{bmatrix}
\]

\[
\gamma = t_n -(n-1)t_1 + \sum_{k=2}^{n-1} t_k
\]
\[
\begin{bmatrix}
1 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{n-1} \\
0 & 1 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & \frac{1}{n-1} \\
0 & 0 & 1 & \ldots & 0 & 0 & 0 & 0 & 0 & \frac{1}{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 0 & 0 & 0 & 0 & \frac{1}{n-1} \\
0 & 0 & 0 & \ldots & 0 & 1 & 0 & 0 & 0 & \frac{1}{n-1} \\
0 & 0 & 0 & \ldots & 0 & 0 & 1 & 0 & 0 & \frac{1}{n-1} \\
0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1 & 0 & \frac{1}{n-1} \\
0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 1 & \frac{1}{n-1} \\
\end{bmatrix}
\begin{bmatrix}
t_1 - t_n + \frac{\gamma}{n-1} \\
t_1 - t_{n-1} + \frac{\gamma}{n-1} \\
t_1 - t_{n-2} + \frac{\gamma}{n-1} \\
\vdots \\
t_1 - t_5 + \frac{\gamma}{n-1} \\
t_1 - t_4 + \frac{\gamma}{n-1} \\
t_1 - t_3 + \frac{\gamma}{n-1} \\
t_1 - t_2 + \frac{\gamma}{n-1} \\
t_1 - t_1 + \frac{\gamma}{n-1} \\
\end{bmatrix}
\]

\[\hat{\alpha}_{n+1} = \alpha\]
\[\hat{\alpha}_i = \begin{cases} 
t_1 - t_n + \frac{\gamma}{n-1} - \frac{\alpha}{n-1} & \text{if } i = 1 \\
t_1 - t_n + \frac{\gamma}{n-1} - \frac{\alpha}{n-1} & \text{if } 1 = 2, \ldots, n-1 \\
\frac{\gamma}{n-1} - \frac{\alpha}{n-1} & \text{if } i = n 
\end{cases}\]

Switch the columns back to original order:

\[\hat{\alpha}_0 = \alpha\]
\[\hat{\alpha}_i = \begin{cases} 
t - t & -\text{if } i = n \\
t_1 - t_n + \frac{\gamma}{n-1} - \frac{\alpha}{n-1} & \text{if } 1 = 2, \ldots, n-1 \\
\frac{\gamma}{n-1} - \frac{\alpha}{n-1} & \text{if } i = 1 
\end{cases}\]
Now the general expression for $x_i$ is more noticeable:

$$u_i = t_1 - t_i + \frac{\sum_{k=1}^{n} t_k - (n-2)t_i}{n-1}$$

$$= t_1 - t_i + \frac{(n-1)t_1 - (n-2)t_i}{n-1} - \frac{\alpha}{n-1} + \sum_{k=1}^{n} t_k - (n-2)t_i$$

$$= \frac{(n-1)t_1 - (n-2)t_i + \sum_{k=1}^{n} t_k - (n-2)t_i}{n-1} - \frac{\alpha}{n-1}$$

$$= \frac{\lambda}{n-1} - t_i - \frac{\alpha}{n-1}$$

where $\lambda = \frac{1}{n-1} \sum_{k=1}^{n} t_k$ is the number of "1" entries in the test vector.

The pseudoinverse will choose the minimum length representation therefore we have to minimize

$$N = \sum_{i=0}^{n} u_i^2$$

with respect to $\alpha$.

$$\frac{\partial N}{\partial \alpha} = \sum_{i=1}^{n} \frac{\lambda}{n-1} - t_i - \frac{\alpha}{n-1} + 2\alpha = 0$$

$$\frac{n a}{(n-1)^2} \frac{\text{nil}}{(n-1)^2} + \frac{\lambda}{n-1} + 2\alpha = 0$$

$$\alpha \left( \frac{n}{(n-1)^2} + 2 \right) = \lambda \left( \frac{1}{(n-1)^2} \right)$$

$$\alpha = \frac{\lambda}{n + 2(n-1)^2}$$
Therefore the activation level will be:

\[
a = 0.22u_0 - 0.22 \sum_{i=1}^{\ell} u_i = 0.22\alpha - 0.22 \sum (\frac{1}{x_i} - t_i \frac{x_i}{n}) = 0.22 \left[ \frac{\lambda}{n + (n-1)^2} \frac{-n\lambda}{n-1} + \frac{n\lambda}{(n-1)n + 2(n-1)^3} \right] = 0.22 \left[ \frac{1}{n + (n-1)^2} \frac{-n}{n-1} + 1 + \frac{n}{(n-1)n + 2(n-1)^3} \right] \lambda = \lambda
\]

Activation is linearly changing with the number of set-bits in the test vector. Even if the classes are very close we can estimate a bias for a nonlinear amplification function which will emphasize the class boundary. According to these results, we should be able to classify any test vector with less than \((n-1)\) "1" bits successfully, without training for them explicitly. This proves our assertion.

With the last improvement, the number of data pairs is reduced from \(T = \sum_{i=1}^{\ell} (2^i - 1)\) to \(T = \sum_{i=1}^{\ell} (1+i)\). With a closer look we see that the improvement is from exponential dependency on \(k\) to linear dependency.

Now we return to the analysis of generalization for the vectors from the difference set \(\hat{P}(S) - P\).

It would be a tedious task to do a similar analysis for vectors with parts in multiple masks. One of the solutions is to follow an experimental verification method. However we prefer to be on the
safe side, so we will devise an algorithm using the tools we have proved so far.

Since we do not train the network for the vectors in the difference set and we did not prove that they result in useful generalizations, we will suspect them all for causing spurious states. Then, we have a concrete definition of these vectors and we can detect the spurious states before they occur.

We will assume that any input vector which has nonzero bits in more than one mask will cause a spurious state. In order to prevent spurious states we will generate a set of vectors from the input vector using masks and propagate them instead of the original vector. We know that vectors embedded totally, in one single mask will not cause spurious states from our analysis. Therefore by this method we will have effectively prohibited spurious state possibility.

The way to use the masks to generate new vectors is as follows:

Let \( t \) be an arbitrary input vector. Generate the vectors:

\[
    t^k = m_k \land t, \quad k = 1, \ldots, n_e
\]

where \( m_k \) is the k-th mask as defined before. Total response for the test vector \( t \) is obtained by:

\[
    y = \sum_{k=1}^{n_e} \text{sig}(Wt^k)
\]

These operations are well-defined because of the properties i-iv of \( S_1, \xi_1, \) and \( \varphi_1, \) and the properties of simple scheme coding.
Since we ruled out the possibility of spurious states, the theory predicts that the recall success should be hundred percent.

4.5. The Algorithm

Now we express the procedure we explained above as an algorithm.

Training:

1-) For each event node in the activity network, collect the representations of \( i_k \)-combination and \((i_k-1)\)-combinations of the fan-in activities in columns of \( X \):

\[
X = [x_{i_k1}^k, x_{(i_k-1)1}^k, \ldots, x_{(i_k-1)(i_k-1)}^k] \quad k=1,\ldots,n_e
\]

2-) Append a column of "1"s to \( X \) to estimate the bias weight values for the nonlinearities which will amplify the class boundaries.

3-) Create the corresponding desired outputs matrix:

\[
D = [\text{rep}(\phi_k), 0, \ldots, 0] \quad k=1,\ldots,n_e
\]

4-) Modify the elements of \( D \) so that the range of the sigmoid is \([0.1, 0.9]\) and then obtain the desired activation values by mapping the modified elements of \( D \) back through the inverse of sigmoid:

\[
E = [\text{sig}^{-1}(\text{rep}(\phi_k)), \text{sig}^{-1}(0.1), \ldots, \text{sig}^{-1}(0.1)] \quad k=1,\ldots,n_e
\]
5-) Calculate the weight matrix:

\[ W = EX^* \]

where \( X^* \) is the pseudo inverse of \( X \).

Testing:

1-) Generate \( n_e \) vectors using the masks, where: \( n_e \) is the number of events:

\[ t^k = m_k \otimes t, \quad k = 1, \ldots, n_e; \quad m_k = \text{rep}(\xi_k) \]

2-) Obtain the total response by:

\[ y = \sum_{k=1}^{n_e} \text{sig}(Wt^k) \]

### 4.6. Experimental Results

We are going to demonstrate the algorithm and the experiment results on the activity network example of the second chapter.

The training set \( X \) and \( D \) can be obtained from the activity network by inspection or from one of the adjacency matrices as follows:

\[ X = \]

Columns 1 through 12

\[
\begin{align*}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{align*}
\]
<table>
<thead>
<tr>
<th>Columns 13 through 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>
Notice that there is a row of ones appended to the X matrix. These are the constant bias inputs that will be used to estimate the locations of nonlinearities.

We have to scale Y to the range [0.1, 0.9], and we have to map it back through inverse of sigmoid to get the desired activations E:

\[
\tilde{y}(i,j) = 0.8 \times y(i,j) + 0.1
\]

\[
e(i,j) = \text{sig}^{-1}(\tilde{y}(i,j))
= \frac{1}{\ln(e^{y(i,j)} - 1)}
\]

\[
E = \\
\begin{array}{cccccccc}
\text{Columns 1 through 7} \\
-0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
-0.2197 & 0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
-0.2197 & -0.2197 & 0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 \\
-0.2197 & -0.2197 & -0.2197 & 0.2197 & -0.2197 & -0.2197 & -0.2197 \\
-0.2197 & -0.2197 & -0.2197 & -0.2197 & 0.2197 & -0.2197 & -0.2197 \\
-0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & 0.2197 & -0.2197 \\
-0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & 0.2197 \\
-0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197 & -0.2197
\end{array}
\]
<table>
<thead>
<tr>
<th>Columns 8 through 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Columns 15 through 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
<tr>
<td>-0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197  -0.2197</td>
</tr>
</tbody>
</table>
Then the weight matrix is just \( W = EX^* \):

\[
W =
\]

Columns 1 through 7

\[
\begin{array}{cccccccc}
0.0000 & 0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 \\
0.4394 & 0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 \\
0.4394 & 0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 \\
0.4394 & 0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 \\
-0.0000 & 0.4394 & -0.0000 & 0.0000 & -0.0000 & -0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.4394 & -0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.4394 & -0.0000 & 0.0000 & 0.0000 & 0.0000 \\
-0.0000 & 0.0000 & -0.0000 & 0.4394 & 0.0000 & -0.0000 & -0.0000 \\
-0.0000 & 0.0000 & -0.0000 & 0.4394 & 0.0000 & -0.0000 & -0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.4394 & -0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.4394 & -0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.4394 & -0.0000 & 0.0000 \\
0.3418 & 0.3418 & 0.3418 & 0.3418 & 0.3418 & 0.1490 & 0.1490 \\
-0.0000 & 0.0000 & -0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.4232 & 0.4232 & 0.4232 & 0.4232 & 0.4232 & 0.1302 & 0.1302 \\
0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 & -0.0000 \\
0.3418 & 0.3418 & 0.3418 & 0.3418 & 0.3418 & 0.1052 & 0.1052 \\
\end{array}
\]

Columns 8 through 14

\[
\begin{array}{cccccccc}
-0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 \\
-0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 \\
-0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 \\
-0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & -0.0000 \\
\end{array}
\]
We can evaluate the performance of training set for the first step by $Y_o = \text{sig}(WX)$:
\[ Yo = \begin{array}{cccccccc}
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.9000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.9854 & 0.2414 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.2278 & 0.0743 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1965 & 0.0787
\end{array}\]

<table>
<thead>
<tr>
<th>Columns 8 through 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.1000 0.1000 0.1000 0.1000 0.1000 0.1000 0.1000</td>
</tr>
<tr>
<td>0.2414 0.2414 0.2414 0.1000 0.2278 0.0638 0.0638</td>
</tr>
</tbody>
</table>
Although this result is not the same as the rescaled version of Y, we can still recover the original Y from the result by using a simple threshold. Therefore the recall rate for the training set is 100 percent.

Observe from the weight matrix that the results close to 0.5 stem from the crosstalk caused by the excitatory weights in the columns corresponding to multiple fan-in nodes. One major advantage of NNets is fault tolerance. Even when some of the weights are destructed the performance doesn't change: drastically.
Sometimes weights with small magnitudes are pruned to save from storage space, making use of this argument. We will use this argument to examine the weight matrix and to improve the results. Observe that rows of the weight matrix with single major entry other than the bias weight at the last column resemble the structure of the activity adjacency matrix. This is a clue to determine which elements could be pruned. We will use the adjacency matrix as a template to prune the weight matrix. We will leave the last column of weight matrix untouched. We can do this by multiplying weight matrix with the activity adjacency matrix elementwise. Resulting weight matrix is:

\[ W = \]

Columns 1 through 7

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.4394 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.4394 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.4394 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.4394 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.4394 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.4394 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.4394 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.1490 & 0.1490 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
### Columns 8 through 14

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- 0.1490 0.1490 0 0 0

\[
\begin{array}{cccccccc}
0.4394 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.2441 & 0 & 0.2441 & 0 & 0.2441 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.4394 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

### Columns 15 through 20

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.5615 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
0 & 0 & 0 & 0 & 0 & -0.6429 \\
0 & 0 & 0 & 0 & 0 & -0.2197 \\
\end{array}
\]
We can test the new weight matrix on the training set again. Now it is possible to see the reduction of the crosstalk in the result:

\[
Y =
\begin{array}{cccccccccc}
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.9000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 & 0.0016 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036 & 0.0036
\end{array}
\]

Columns 8 through 14

\[
\begin{array}{cccccccccc}
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\
0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000
\end{array}
\]
Before we test the system over all possible test inputs we take one more step to change the weights matrix. Since we pruned most of the sonnection weights we gained a lot in terms of storage space. However we can still gain some more. The remaining weights are real numbers. The binary nature of the problem suggests that
integers or rational numbers should be able to solve it. Also observe
that entries on a single row which are other than zero have the same
magnitude. Therefore we can replace these weights by "1" and scale
the bias weight suitably. In fact real numbers are just a left over of
crosstalk which was present in the initial weight matrix. What is the
suitable way to scale the bias weights? These weights locate the
class boundary between n-combinations and (n-1)-combinations of
the fan-in set. Therefore we have to find out from the pruned
weight matrix with real number entries that how many of a columns'
nonzero entries are required to exceed this threshold. Then we
merely set the value to this number. Now, at the output we can use
thresholding nonlinearity directly instead of sigmoid followed by
threshold because the effect of sigmoid will be absorbed in threshold.
If we define the discontinuity point of the threshold to be included to
the right portion of the function than the above definition the choice
of bias weight works just fine. To demonstrate this we modify our
weight matrix to have only integer weights:

\[ W = \]

Columns 1 through 12

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
We tested this weight matrix for all possible test inputs exhaustively and compared the results with the activity network using the adjacency matrix. *Matlab* program used for this is given in the appendix. The recall rate is 100 percent. We obtained same

| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Columns 13 through 20 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
accuracy for activity networks with \textit{upto 50} activities which is a moderate number.
5. **FUZZY ASSOCIATIVE MEMORIES FOR RESOLVING RESOURCE CONSTRAINTS**

5.1. Fuzzy Associative Memories

Fuzzy Associative Memories (FAMs) map fuzzy sets to fuzzy sets, a unit cube corresponding to the power set of a universe of discourse (or ground set) to another unit cube. This abstraction provides an alternative to propositional and predicate calculus reasoning techniques used in artificial-intelligence (AI) expert systems, allowing us to reason with sets instead of propositions. The AI framework is symbolic and one-dimensional, with usually only bivalent expert "rules", whereas the fuzzy-set framework is numerical and multivalued. Both frameworks can encode structured knowledge in linguistic form, but the fuzzy approach translates the structured knowledge into a flexible numerical framework and processes it in a manner that resembles neural network processing. The numerical framework also allows us to adaptively infer and modify fuzzy systems. FAMs may also be preferable over neural networks, since they have a trail of inference by which the outputs of the system can be justified.

The simplest FAM system consists of a single association such as \((A, B)\) where \(A\) and \(B\) are linguistic variables. Examples for linguistic variables are small, high, etc. A universe of discourse is a
domain for linguistic variables. In general, FAM system consists of several such associations. For simplicity assume FAM has a single association \((A, B)\). For example, assume that the universe of discourse \(X\) for the fuzzy set \(A\) is the amount of slack of an activity and the universe of discourse \(Y\) for the fuzzy set \(B\) is the priority of that activity for using a type of resource. Both universes may be considered as subsets of the real line. In this case, they are continuously infinite sets. The fuzzy sets \(A\) and \(B\) are multivalued or fuzzy subsets of \(X\) and \(Y\). \(A\) and \(B\) define the membership functions \(m_A\) and \(m_B\) that map the elements \(x\) of \(X\) and \(y\) of \(Y\) to degrees of membership in \([0, 1]\). The membership values indicate how much \(x\) belongs to \(A\), and how much \(y\) belongs to \(B\). We describe this with the abstract functions \(m_A: X \rightarrow [0, 1]\) and \(m_B: Y \rightarrow [0, 1]\). We can view the membership functions in graphs where one axis denotes the variable for universe of discourse and the other axis denotes the membership values. This is demonstrated in figure 5.1. We can quantize the universe \(X\) to the \(n\) numerical variables \(x_1, x_2, \ldots, x_n\) and the universe \(Y\) to the \(p\) numerical variables \(y_1, y_2, \ldots, y_p\). Elements \(x_i\) and \(y_j\) belong to the ground sets \(X = \{x_1, x_2, \ldots, x_n\}\) and \(Y = \{y_1, y_2, \ldots, y_p\}\), respectively. In this case the unit cube corresponding to the universe of discourse \(X\) is \(n\)-dimensional, and the unit cube corresponding to the universe of discourse \(Y\) is \(p\)-dimensional. The membership functions are the sampled versions of the ones displayed in figure 5.1.1 for the continuous case. We can represent the fuzzy sets \(A\) and \(B\) as vectors \(A = (a_1, a_2, \ldots, a_n)\) and \(B = (b_1, b_2, \ldots, b_p)\), where \(a_i = m_A(x_i)\), and \(b_j = m_B(y_j)\).
We define vector-matrix multiplication and summation using min and max operators in a way that resembles the classical definitions. The pairwise multiplication is replaced by pairwise minima, and the column (row) sums are replaced by column (row) maxima. This operation is called max-min decomposition and denoted by the operator "o". Then the definition for $AoM=B$ where $M$ is $n$-by-$p$, is

$$b_j = \max_{i \leq n} \min(a_i, m_u)$$

The association $(A,B)$ is stored in the FAM using the matrix $M$ defined by correlation-minimum encoding as

$$m_u = \min(a_i, b_j)$$
or in vector notation:

\[ M = A' \ominus B \]

M can be expressed in the following ways also:

\[
M = \left[ \begin{array}{c}
\hat{a} \land B \\
\vdots \\
\hat{a} \land B
\end{array} \right] = \left[ b_1 \land A' \land A' \land \ldots \land b_m \land A' \right]
\]

where the cap operator denotes the *pairwise* minimum:

\[ a_i \land b_j = \min(a_i, b_j) \]

and

\[ a_i \land B = (a_i \land b_1, \ldots, a_i \land b_m). \]

Recall is dictated by the bidirectional FAM theorem for correlation-minimum encoding by Kosko \[\text{[Kosko92]}\]:

If \( M = A \odot B \), then

(i) \( A_0 M = B \) iff \( H(A) \geq H(B) \)

(ii) \( B_0 M = A \) iff \( H(B) \geq H(A) \)

(iii) \( A' \odot M \subset B \) for any \( A' \)

(iv) \( B' \odot M \subset A \) for any \( B' \)

where \( H(A) \) is the height of the fuzzy set \( A \) defined as the maximum fit value of \( A \):

\[ H(A) = \max_{i \in \text{supp}(a_i)} a_i \]
When there are m FAM rules or associations \((A_1, B_1), ..., (A_m, B_m)\), instead of adding the matrices corresponding to each association, which would be defined as

\[ M = \max_{1 \leq k \leq m} M_k \]

we add the recalled vectors

\[ B = \sum w_k B'_k \]

where \( B'_k = A_k M k \), and \( w_k \) are the relative weights of rules on overall decision.

This means that the matrices are kept separately. Although storage space is consumed, this method provides an audit trail for the inference procedure and avoids crosstalk. FAM system architecture is shown in figure 5.1.2.

![FAM system architecture](image)

Figure 5.1.2. FAM system architecture.

Since actual systems present only exact inputs we prefer using Binary Input-Output FAMs (BIOFAMs). BIOFAM treats antecedent fuzzy sets as propositions with fuzzy truth values. Also, in actual systems, the output of the FAM is expected to be a single number.
instead of a fuzzy set. This is taken care of using defuzzification.
Again, in reality, combinations of antecedents are used instead of simple single antecedents. For this, the compound rule is split into simple rules.

For example, the compound rule "if A and B, then C" represented by \((A,B;C)\) is split in the following way:
Each antecedent is stored separately:
\[
M_{AC} = A \circ o C \\
M_{BC} = B \circ o C
\]
Recall for each antecedent is also separate:
\[
C_A = A \circ o M_{AC} \\
C_B = B \circ o M_{BC}
\]
Then the recalled vectors are combined in the same fashion as the antecedents which caused recall them:
\[
F(A',B') = [A \circ o M_{AC}] \cap [B \circ o M_{BC}] \\
= C_A' \cap C_B' \\
= C
\]
C' is the overall recalled fuzzy set. Notice that since the antecedents were combined with AND conjunction, the recalled sets are combined with intersection. To generalize this, we can tabulate the process as follows:

\[
\begin{align*}
[A \rightarrow (B \ OR \ c)] & \rightarrow [(A \rightarrow B) \ OR \ (A \rightarrow c)] \\
[A \rightarrow (B \ AND \ C)] & \rightarrow [(A \rightarrow B) \ AND \ (A \rightarrow c)] \\
[(A \ OR \ B) \rightarrow C] & \rightarrow [(A \rightarrow C) \ OR \ (B \rightarrow C)] \\
[(A \ AND \ B) \rightarrow C] & \rightarrow [(A \rightarrow C) \ AND \ (B \rightarrow C)]
\end{align*}
\]
We now examine the BIOFAM case where the input is exact.

Let \( \mathbf{1}_x \) = \((0.010...0)\) denote the n-dimensional bit vector with i-th element 1 and all other 0. \( \mathbf{1}_x \) equals the i-th row of the n-dimensional identity matrix. In the continuous case \( \mathbf{1}_x \) denotes a delta pulse in a convolution integral. Define \( \mathbf{1}_y \) and \( \mathbf{1}_z \) similarly. When the inputs are \( \mathbf{1}_x \) and \( \mathbf{1}_y \) instead of \( A' \) and \( B' \), recall occurs as follows:

\[
F(X_i, X_j) = F(I_i, I_j) = [I_i \cdot \mathbf{oM_{AC}}] \cap [I_j \cdot \mathbf{oM_{BC}}] \\
= (a_i \land C) \cap (b_j \land C) \\
= \min(a_i, b_j) \land C \\
= \min(a_i, b_j) \land m_c(z) \quad \forall z \in Z
\]

The output defuzzification to express the output fuzzy set as a single number is required as a last step. Simplest defuzzification scheme chooses the element \( y_{\text{max}} \) that has maximal membership in the output 'fuzzy set C:

\[
m_c(y_{\text{max}}) = \max_{i \in I} m_c(y_i)
\]

This is basically similar to maximum likelihood estimation. However, above description is ambiguous if the fuzzy set has a flat top at the maximum region. In this case an alternative definition, fuzzy centroid, can be used:

\[
\text{centroid}(C) = \frac{\sum_i y_i m_c(y_i)}{\sum_i m_c(y_i)}
\]

in the discrete case and

\[
\text{centroid}(c) = \frac{\int y \cdot m_c(y) dy}{\int m_c(y) dy}
\]

in the continuous case.
5.2. Fine-Tuning The FAM

Kosko asserts that one advantage of FAMs over NNets is they don't need training [Kosko92] FAMs can be directly generated from expert rules if they are available. Kosko himself uses some rules of thumbs. One of these is to have approximately 25 percent overlap between sets. Too little overlap tends to resemble bivalent expert rules of AI. Too much overlap blurs the distinction between the fuzzy set values. Another one is to keep the fuzzy set membership functions corresponding to the equilibrium conditions narrow, and expand the widths of sets towards the ends of the state variable range. This gives fault tolerance to the system. Even if there is an error in the most important rule for stability, the equilibrium condition, the system can correct the error when the corresponding membership function is narrow. Then the system will be piecewise stable. Trapezoidal or triangular fuzzy set membership functions are used widely. Because of the fault tolerance of the FAM they work as good as arbitrary membership functions, and since they are easily interpolated they are convenient. Then, the parameters which can be controlled are the upper and lower base widths and overlap.

Kosko uses competitive learning algorithms with historical data of the system under investigation, to find the optimal number of rules, the rules themselves and their weights. When a differential competitive learning algorithm can be used to estimate the rules. The cluster centers are the FAM rules. The number of vectors clustering
around a cluster center, divided by the total number of vectors give the weights of the rules.

If historical data about the system is available, using adaptive techniques is an alternative for using expert knowledge which may not be at hand all the time.

Lin and Lee use a hybrid learning algorithm [LinLee91]. The first phase is a self-organizing method to estimate the locations and to a degree the shapes of membership functions. They start with the assumption that the membership functions are gaussian. They estimate the locations of the membership functions, or the means of the gaussians by a clustering algorithm. They use Kohonen's feature-maps algorithm [Kohone88] for this purpose. The standard deviations of gaussians are taken as the distance to the first nearest neighborhood, determining the total shape of each membership function. Then they use a competitive learning algorithm to determine the rules as Kosko does. The second phase is supervised learning to fine-tune the variables whose initial values are estimated as above.

The gaussian-shaped membership function assumption is a little better than the trapezoidal assumption of Kosko. Since gaussian function is differentiable, it allows a fine-tuning algorithm to be applied which uses the gradient descent method. Another assumption is in taking the first nearest neighbor distance as the standard deviation of the gaussian function. This is similar to 25 percent overlap rule Kosko uses.
Cho and Ersoy propose a method based on the distribution of samples in the historical data as an alternative technique for deriving the membership functions of fuzzy sets [ChoErs92]. For each set of samples belonging to a class, Cho and Ersoy construct an histogram. Using the histogram he defines the membership function as follows:

\[ m_{A_i}(x_j) = \frac{1}{N_{ci}} \sum_{k=1}^{I} \min(N_j, N_k) \]

where \( m_{A_i}(x_j) \) is the membership of \( x_j \) in \( A_i \), \( N_j \) and \( N_k \) are the number of samples in the j-th and k-th intervals, respectively, \( I \) is the number of intervals, and \( N_{ci} \) is the total number of samples in class i. After determining fuzzy sets, he computes a similarity function using euclidean or Hamming distances as measure, and merges the sets that are close. The membership function is redefined for the sets which are merged together. In this method, the degree of overlap between sets is controlled by the number of samples classified in more than one class.

5.3. Resolving the Resource Constraints with BIOFAM

We will again demonstrate the implementation with the example activity network of the second chapter. We will implement a BIOFAM with continuous universe of discourses. MINSLK and SA rules will be used as two ground sets. Therefore slack (or total float) and activity duration will be our fuzzy state variables. The fuzzy control variable, or the output, will be the priority of an activity for a
given resource. Using figure 2.2.1 we can tabulate the values for these state variables which would result from a CPM analysis as shown in Table 5.3.1 below.

Table 5.3.1. Durations and total floats of the activities of figure 2.2.1.

<table>
<thead>
<tr>
<th>activity no</th>
<th>duration</th>
<th>total float</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Histograms corresponding to these state variables are given in figure 5.3.1. If we had more data we would like to use the method of Cho and Ersoy, since it is more objective in choice of shapes and locations of fuzzy sets. However, CPM doesn't generate too much data, and we will prefer Kosko's method which is easier to implement when data is scarce.
The range of durations is from 0 to 9. Since the heuristic to be implemented is "choose shortest activity", the equilibrium case will be located around zero. Clustering in histogram tells that 3 fuzzy sets will be enough. We name them as SHORT, MEDIUM, and LONG. Membership functions are demonstrated in figure 5.3.2. We obtained these functions from the inspection of histogram and 25 percent overlap rule. The range of slacks is from 0 to 15. Similarly we define three fuzzy sets as LARGE, MEDIUM and SMALL. Membership functions are shown in figure 5.3.3, obtained by the same method. Equilibrium is on the SMALL side, since the rule is "choose the
activity with the minimum slack". We define three fuzzy sets as LOW, MEDIUM and HIGH for the control variable, priority, over the continuous universe of discourse from 0 to 1. The membership functions are shown in figure 5.3.4.

Figure 5.3.2. Fuzzy sets over ground set "durations" and the membership functions.

Figure 5.3.3. Fuzzy sets over ground set "slacks" and the membership functions.
The FAM rules will be triples such as \((\text{SHORT}, \text{SMALL}; \text{HIGH})\) which would read as "If the duration of the activity is SHORT and the slack is SMALL, then give the activity a HIGH priority for resource." We can show the rules on the cartesian space of the state variables duration and slack. We don't have to define a rule for each pair.

If we had historical data, we would define a rule for those pairs which occur relatively frequently. We can alternatively choose to define rules for the pairs which are close to equilibrium cases, and rely on the system's ability to restore equilibrium quickly. This would be possible since we chose the sets for the equillibrium cases narrow. The rules we chose are tabulated in table 5.3.2.
Table 5.3.2. Priority values for FAM rules. (X : no rule defined for this pair.)

<table>
<thead>
<tr>
<th>slack</th>
<th>duration</th>
<th>SHORT</th>
<th>MEDIUM</th>
<th>LONG</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>LOW</td>
<td></td>
</tr>
<tr>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>LARGE</td>
<td>LOW</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Next we will see how the input \((\text{duration,slack}) = (1,5)\) will get processed. The rules that apply to this pair will be \((\text{SHORT,SMALL;HIGH})\) and \((\text{SHORT,MEDIUM;MEDIUM})\). So the output fuzzy set will be:

\[
F(1,5) = \left[ m_{\text{SHORT}}(1) \land \text{HIGH} \right] \cap \left[ m_{\text{SMALL}}(5) \land \text{HIGH} \right] \\
+ \left[ m_{\text{SHORT}}(1) \land \text{MEDIUM} \right] \cap \left[ m_{\text{MEDIUM}}(5) \land \text{MEDIUM} \right] \\
= \min(m_{\text{SHORT}}(1), m_{\text{SMALL}}(5)) \land \text{HIGH} \\
+ \min(m_{\text{SHORT}}(1), m_{\text{MEDIUM}}(5)) \land \text{MEDIUM} \\
= \min(0.75, 0.20) \land m_{\text{HIGH}}(z) \\
+ \min(0.75, 0.50) \land m_{\text{MEDIUM}}(z) \\
= 0.2 \land m_{\text{HIGH}}(z) + 0.5 \land m_{\text{MEDIUM}}(z)
\]

To get the resulting priority we defuzzify the output:

\[
\text{priority} = \frac{\int_0^1 [0.2 \land m_{\text{HIGH}}(z) + 0.5 \land m_{\text{MEDIUM}}(z)] \, dz}{\int_0^1 [0.2 \land m_{\text{HIGH}}(z) + 0.5 \land m_{\text{MEDIUM}}(z)] \, dz} = 0.5
\]

This whole process is demonstrated in figure 5.3.5 below.
Figure 5.3.5. Operation of BIOFAM.

Note that the priorities provided by the FAM are real numbers. This is good enough to make an ordering of activities.

The rest of the process is taken care of by the third part of simulation model which includes everything that is not included in activity network part or in determining the priorities using
heuristics. This part allocates the available resources to pending activities in the order of priorities assigned by the FAM, until the skyline of resources are reached. It puts the activities which did not receive service from a resource to hold status to evaluate its request again when resources become available.
6. CONCLUSIONS

Although simulation provides suboptimal solutions to operations planning problems, it has a greater flexibility in modeling systems. This gives simulation an advantage over programming techniques which give exact solutions to operations planning problems, but also are limited in the range of problems they can be applied to in practice. Resource constrained operations planning problems are especially hard for programming approaches. Quality of solution provided by the simulation method can be increased significantly if an internal decision mechanism such as heuristic methods is added. For the resource constrained operations planning problem, best solution can be obtained for upto 95 percent of the trials.

A large number of problems solved by simulation can be partitioned into three major parts: (1) activity network, (2) decision mechanism for resolving resource constraints, and (3) simulation engine which includes everything that is not included in (1) and (2). Instead of trying to model a complete system's input-output relations by a single NNet, we use the same partitioning above. This helps in building a more accurate model of the system under investigation.
We obtain some advantages by implementing part (1) and (2) in NNet and FAM technologies, respectively. NNet and FAM paradigms are implementable in parallel technology, and they operate fast. They can be partitioned in very fine granularity if the system they are implemented on allows.

We achieve one hundred percent recall success rate with the NNet designed in chapter four. The FAM designed in chapter five allows us to combine any number of heuristics with adjustable weights. It is already shown that combination of four to five heuristic rules with unit weight can increase the quality of solution [Boctor90]. This success in the parts of the model gets reflected to the results of simulation.

The NNet designed uses a small number of integer valued connections. This decreases the amount of space needed for storage significantly when compared to massively connected NNets with decimal weights. NNet doesn't forget the old data, so it helps retain the corporate memory. The structure of the NNet provides a good outline of the structure of the system it models. And FAM provides an inference trail that can be adopted in operation plan's execution should a deviation from the plan occur.

Further research includes relaxing the assumption on graphs. Cycles in graphs should be allowed since a large number of systems have cycles in their activity networks. Renumbering of nodes using the algorithm of chapter two is cumbersome when too many nodes are added. Node numbers are only needed for our reference to the
activities. Otherwise, the **NNet** is trained on subsets of the power set of activities. Training from reachability matrices instead of adjacency matrices is desirable. The latter can be obtained from the former if certain assumptions are made. This is demonstrated by Sage [Sage77]. However, then we will have to relax those assumptions too.

Performance should be tested on larger systems. **Upto** now we have tried 50-task problems. When the number of tasks increase, computing the pseudoinverse by inverting the gram matrix get harder. Two possible solutions are using sequential least squares by making multiple passes over the available data to increase the accuracy of the approximation to the inverse of gram matrix, and partitioning the system down to manageable pieces. We have to test the model at real problems. Also on-line adaptation methods for FAM and **NNet** parts would enhance the configurability of the design.
REFERENCES


[BrMeSh64] - Brand, J. D., Meyer, W. L., and Shaffer, L. R.: The resource scheduling problem in construction", Civil engineering studies report no. 5, Department of Civil Engineering, University of Illinois, Urbana, IL.


APPENDIX
APPENDIX

The **Matlab** programs used for testing **NNets** of chapter four are given below. The first one uses masks. The second program does not use any mask.

1.)

load AN;
load X;
load Y;
load M;

\[
[Nm \ Na] = \text{size}(M);
[nT Na] = \text{size}(Y);
\]

\[
Y = Y.*0.8 + 0.1;
\]

s = 10;

\[
A = \log\left(\frac{Y}{1-Y}\right) \div s;
\]

\[
W = \text{pinv}(X)A;
\]

\[
Ao = X*W;
\]

\[
Yo = 1 \div (1 + \exp(-s.*Ao));
\]

error = 0;

for i=1:2^Na
  \( k = 1; \)
  for j=1:Na
    \( k = k*2; \)
    \( T(j) = \text{rem}(i,k); \)
  end

\[
A1 = \text{zeros}(1,Na);
\]

for i=1:Nm
  \( A1 = A1 + (T.*M(i,:))*W; \)
\[ Y_1 = \frac{1}{1 + \exp(-s \cdot A_1)}; \]
for \( i = 1:Na \)
  if \( (Y_1(i) \geq 0.5) \) \( Y_1(i) = 1; \) else \( Y_1(i) = 0; \) end
end

\[ Y_2 = T \cdot AN(1:Na,:); \]
for \( i = 1:Na \)
  if \( (Y_2(i) \geq A_2(Na+1,i)) \) \( Y_2(i) = 1; \) else \( Y_2(i) = 0; \) end
end
if \( \sum(\text{abs}((y_1-y_2))) = 0 \) error = error + 1; end

2.)
load AN;
load X;
load Y;
load M;

[Nm Na] = size(M);
[nT Na] = size(Y);

\[ Y = Y.*0.8 + 0.1; \]
\[ s = 10; \]
\[ A = \log( \frac{Y}{1-Y} \) \] / s;

\[ W = \text{pinv}(X) \cdot A; \]
\[ A_0 = X \cdot W; \]
\[ Y_0 = \frac{1}{1 + \exp(-s \cdot A_0)}; \]

error = 0;
for \( i = 1:2^\text{Na} \)
  \[ k = 1; \]
for j=1:Na
    k = k*2;
    T(j) = rem(i,k);
end

A1 = T*W;
Y1 = 1 ./ ( 1 + exp(-s.*A1) );
for i=1:Na
    if (Y1(i) >= 0.5) Y1(i) = 1;
    else Y1(i) = 0;
end

Y2 = T*AN(1:Na,:);
for i=1:Na
    if (Y2(i) >= AN(Na+1,i)) Y2(i) = 1;
    else Y2(i) = 0;
end
end
if ( sum(abs(y1-y2)) ~= 0 )
    error = error + 1;
end
end