A Neural Network Simulation Package in CLIPS

Himanshu Bhatnagar, Patrick D. Krolak, Brenda J. McGee, John Coleman.

Center for Productivity Enhancement, University of Lowell, Lowell, Ma. 01853

ABSTRACT

The intrinsic similarity between the firing of a rule and the firing of a neuron has been captured, in this research, to provide a neural network development system within an existing production system (CLIPS). A very important by-product of this research has been the emergence of an integrated technique of using rule based systems in conjunction with the neural networks to solve complex problems. The system provides a tool kit for an integrated use of the two techniques and is also extendible to accommodate other AI techniques like the semantic networks, connectionist networks, and even the petri nets. This integrated technique can be very useful in solving complex AI problems.

1. INTRODUCTION

Direct hardware implementation of Neural Networks is not always easy and hence there is a need for simulating them through computer software. Early examples of software simulation models can be found in [1] and [2]. These and the other simulation models primarily simulate the neural states, neural architectures and connection strengths, and implement the tools to manipulate them. Several learning techniques (rules) have been proposed in the Neural Network literature, one of them being the generalized delta rule (or Back Propagation)[3]. Our first level goal is to provide a more efficient package, in CLIPS, for simulating neural networks employing back propagation, together with expert systems.

CLIPS is an expert system shell developed by NASA [4], which provides a LISP like interface and allows both forward and backward chaining. The production rules, under forward chaining, have facts on the lhs and action commands on the rhs. When facts, in the facts database, match the lhs of any rule that rule fires, possibly causing assertion of more facts and hence firing of other rules. In a binary neural network, a neuron fires when its activation has exceeded its threshold value. There is an inherent similarity in the way rules fire in an expert system and the way neurons fire in a Neural Network, suggesting the modeling of one in terms of the other, and hence CLIPS can prove to be a very effective simulation tool for Neural Network modeling. We, at the Center for Productivity Enhancement, University of Lowell, have developed a shell called Neural CLIPS, or N-CLIPS which allows Neural Network Simulations to be built, tested and implemented along with regular expert systems. N-CLIPS provides a common environment for development, implementation and operation of two competing and radically different artificial intelligence techniques : the C Language Integrated Production System (CLIPS) for writing expert systems and a Neural Network system. These systems can either operate independently to solve different classes of artificial intelligence problems or can cooperate to help solve much bigger AI problems [9]. In [6] Rabelo has shown the usefulness of combining the neural networks and the expert systems. Knowledge representation, acquisition and manipulation, decision making and decision support are the major characteristics of these techniques and hence when they are used together they can share knowledge and can share the decision making process itself.

To further emphasize the importance of such a common platform we are using it to model a traffic control system for mobile robots operating the Material Handling System of a Flexible Manufacturing System based factory [7]. The (simulated) mobile robots have on-board neural networks which work together with expert system modules to guide them through the factory floor without collisions and with minimum delays. Since CLIPS provides an excellent interface with C, these expert system rules can interact with other processes and also interact with different types of peripheral hardware [5].

The next section provides a brief description of the terms relevant to neural networks, followed by a survey of the features common to currently available simulation packages. The need for integrating AI techniques is discussed next followed by a description of N-CLIPS. The last section gives a detailed explanation of the system developed.

2. ARTIFICIAL NEURAL NETWORKS

2.1 Definitions

For our purposes a **neural network** is a densely connected, possibly layered, network of simple processing units (**neurons**). The connections, known as **synapses,** are weighted links between two such units where the **weight** of a link is modifiable, and determines what fraction of the signal, between the two units, is actually passed. A negative weight usually signifies an **inhibitory link**(synapse) which causes an inhibitory effect on the firing of a post-synaptic neuron. A positive weight usually signifies a **excitatory link** which excites the neuron to which it is connected.

Neurons, in the network may be classified into three types depending on the roles they play. They are either **input neurons** (input layer), **output neurons** (output layer) or **hidden neuron** (hidden layer) depending on whether they accept input from outside world, provide an output to the outside world or receive input from units within the system and generate output for the units within the system. Processing within a neuron may be divided into three stages : a) determination of net input to the neuron ; b) determination of neural state (an **activation function** associated with a neuron determines the state); and c) determination of the neural output (an **output function** determines the final output value).

2.2 Learning

The two major learning paradigms available currently are: generalized delta rule (GDR) or back propagation [3] and its variations for both feed forward and **recurrent networks**[16], and **hebbian learning**, with its sophisticated variants (by which we mean to include methods employed in Bi-directional Associative memories and other associative memory models) [10][17][18][19] [20].

2.3 Generalized Delta Rule

In the initial phase of our work we have focused on the GDR as applied to feed forward networks. In this approach a set of patterns is repeatedly presented at the input layer of a multilayered network. The output pattern generated is compared with a target pattern. The difference is propagated back and is reflected as a change in the weights of the links, all the while minimizing a global energy function (**mean squared error** function). The difference or the **delta** is

used to modify the weights of links between neurons. This process is repeated till the actual pattern is within a close range of the target pattern, for a particular input pattern. This is done for each input pattern.

3. EXISTING SIMULATION PACKAGES

A brief survey of most of the commercial neural network simulation and development packages reveals the following characteristics :

* A strong user-interface : Pop-up menus within a windowing environment, a file system and interface with major database systems for I/O.

* Types of Learning Paradigms supported : All major learning paradigms along with their variations.

* Capability for Customizing and designing user-specified Neural Nets : Ranges from just setting up of network parameters to script based design of neural networks.

* Debugging & Interaction tools : On-line graphical editing of a neural network; pausing, restarting and saving snap shots of neural nets during different states of their operation: displaying weight change, delta change, noise and a host of other features.

The different information processing paradigms are particularly well suited for the problem domain in which they evolved. However, when addressing classes of problems that span more than one domain an integrated approach seems attractive. This approach involves several different AI techniques. The inter-relationships of these techniques is still not well understood and there is a need to study their interaction with each other. None of the systems available today have the capability of providing a common platform to investigate these 'inter-relationships'. In N-CLIPS we provide a common playing ground for at least two of these, with the capability of extensions to accommodate others.

4. WHY CLIPS ?

By extending CLIPS to accommodate neural networks, semantic networks, connectionist networks and other knowledge representation techniques, we, will have a tool to understand their complex inter-relationships and the mapping of one technique into another. In real life systems we need the precision of expert systems, the localized representation of semantic networks and the flexibility of neural networks all encompassed into one. This is so because each of these techniques have strengths which compliment the weaknesses of the other. The brittleness of expert systems can be supplemented with the plasticity of neural networks on one hand and the lack of precision of neural networks can be substituted by precise rules and facts. Adding new knowledge to an expert system is quick (as a new rule) but its interaction with the existing rules can be of a conflicting nature. On the other hand adding a new pattern to a neural network takes a long time but can be made to interfere minimally with the old patterns. On a factory floor, new situations can be quickly learned by plugging in temporary rules. However, over a period of time, these rules get to be unmanageable and redundant and have to be trimmed. They can be collectively mapped into a neural network which could iron out the conflicting rules, and once trained it can be mapped back to a more parsimonious set of rules. To illustrate this further, assume a set of rules which do not trigger each other. The combinatorial arrangement of the

union of facts on the lhs of these rules and the actions on the rhs can be translated to the input and the output patterns of a back propagation neural network (BPNN). Out of the available output patterns the ones actually needed can be selected without difficulty. Then by applying the inverse mapping technique proposed by Williams [11] where the input values (at the input layer) instead of the weights are modified via back propagation of error, the neural network can be converted back into an expert systems. Of course, a major problem to be considered in this process is that of knowledge representation since patterns must be translated into facts. In addition there may be many-to-one mappings that are dependent upon initial states of the system.

Sometimes, at a higher level of design the localized representation of a problem can be done through semantic networks and the rest as expert systems and neural networks . For example the higher level path planning of mobile robots on a factory floor can be done using semantic networks, while the low level path planning and traffic control can be done by expert systems which in turn depend on neural networks for decision support. As can be seen all three models will need to communicate with each other. CLIPS allows that via rules and facts, moreso because all of these techniques shall have rules and facts as their building blocks.

Another example would be the cooperative use of multiple neural networks for mortgage underwriting and industrial parts Inspections [13][14][15]. In [13] the system is a collection of nine coupled sub-networks have three sub-networks acting as 'experts' and their cooperative effect helps in validating the confidence level of the decisions made by the whole system.

The major functions which were added to the existing CLIPS code have been briefly explained in Appendix A. The engine for neural networks manipulates its own data structure but eventually uses clips' agenda and fact lists to let the clips execute the neural network. The functions listed in the appendix are driver, nassert, add_nfact, ncompare, ndrive, nretract. PCLIPS [8], a distributed version of CLIPS has also been developed at the university.

5. N-CLIPS

This shell provides an object oriented approach to problem solving in the neural network and fuzzy logic domain and at the same time maintains the integrity of the CLIPS production system. The expert systems and sub-systems can be written as rules and facts while a neural network is represented as a collection of objects and a set of actions to be performed on them. It provides well known neural network learning paradigms as objects which the user can use to map their problems onto or use them as subsystems of more complex user-designed neural networks. Users can also build their own variations of the existing paradigms and can also create their own learning rules and models within the given environment. A library of functions for creating and editing neural network objects like neurons, synapses, activation functions and layers is made available to the user. The *ntrain* and *nrun* functions are a collection of rules linked with facts which can be invoked to train a neural network or execute it. The rules and facts making up the expert systems are written in the same way as in regular CLIPS. At the lowest level of expert system-neural network communication the two systems interact via rules and facts. However, at a higher level, complex but abstract interaction is possible. For example the neural network actions, composite and primitive, can be written as a set of rules linked with facts while an expert system can spawn off a neural network to extract useful information from available fuzzy or smudged knowledge. This system can also be used as a first level tutor for

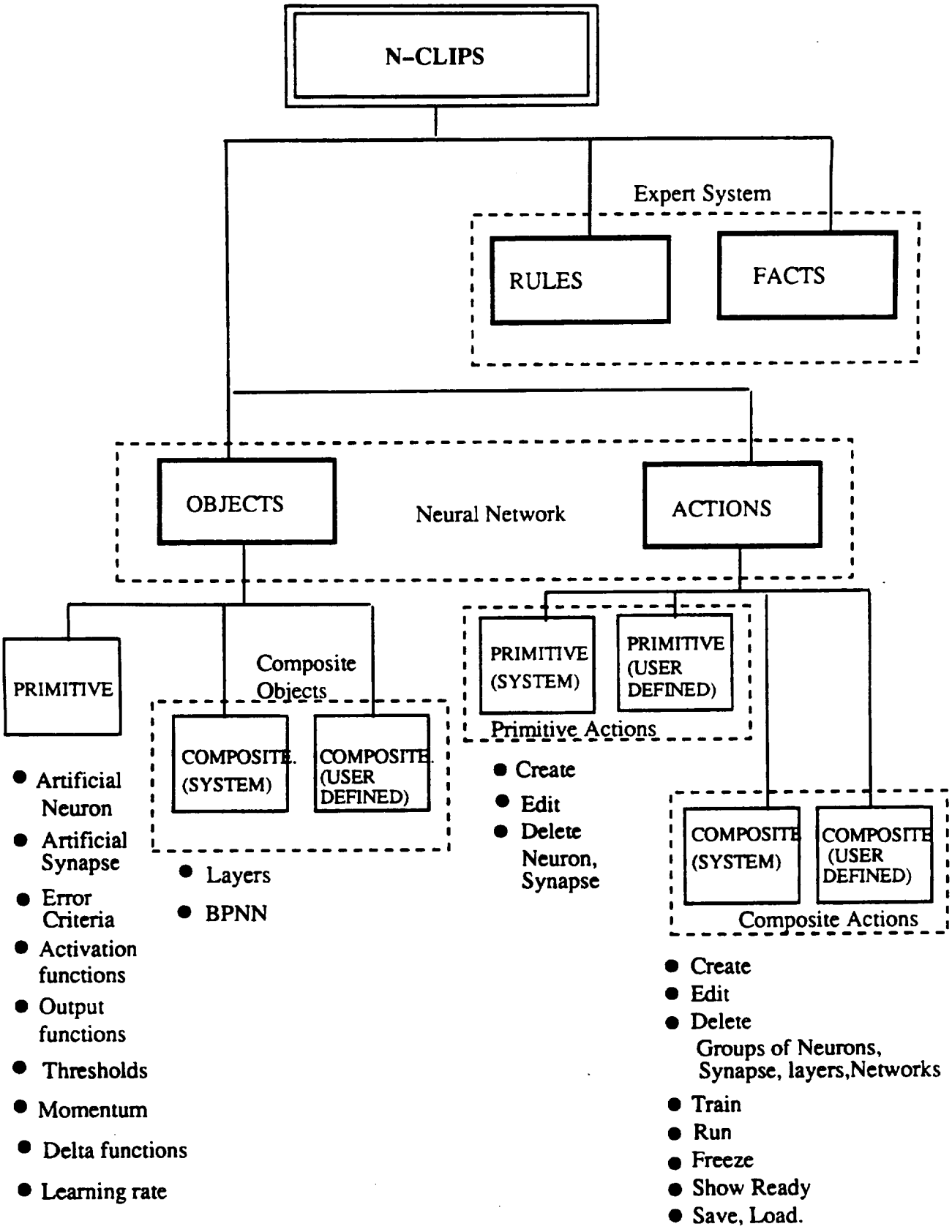

fig 1 : N-CLIPS : A hierarchical description

understanding basic existing models. CLIPSs' capability to interface with other languages viz. C, Ada is exploited for a graphical (X-Windows and/or Motif) user-interface and a file-system interface for saving snap shots and networks themselves. In this system the following graphics user-interface is available :

* Neural Network interconnection diagram.
* CLIPS rules interconnection diagram for seeing which rules fire which other rules and on what basis.
* Mouse interface with the Neural Network diagram.
* 'Click-on-connection-for-weight-change' graphical facility.
* Change of color if a node fires.
* X-Windows link editor.
* X-Windows weight editor.

The file system interface allows saving and loading of neural networks via save_nn() and load_nn() functions, at any instance.

6. SYSTEM DESCRIPTION

6.1 OBJECTS (Primitive)

6.1.1 Artificial Neuron

An artificial neuron is basically of three types i.e. Input, Output and Hidden. Its major characteristics (for back propagation) are an identifying number, layer number, an activation and output function, threshold value and its type. These parameters could be either passed to a C function call or through a template invoked from the CLIPS interpreter. After the parameters of a neuron are accepted from the user they are encoded as a special rule in a string which is then compiled and loaded into the network. These parameters could be edited and a complete neuron deleted at any given instance. Internally in CLIPS the specifications of a neuron are also stored within a data structure (see fig. 2). Any modification of a neuron's specifications are automatically reflected in the data structures and the associated rule. A deleted neuron will also result in deletion of all the connected links.

The composition of the special rule (for back propagation only) is as follows :

```
(defrule artneu#
    ? neu <- (neuron # layer # ready to fire)
=>
      (nretract ? neu)
      (propagate layer #)
      (calculate_delta layer #)
      (change_weights from layer # to layer #)
)
```

On the rhs the function propagate(), propagates the output signal to the next layer neurons after duly multiplying it by the strength of the connection of the links. The next function calculate_delta(), calculates the deltas based on the error signal propagated by the succeeding layer and stores them in the data structures. Finally, the change_weights() function changes weights based on the calculated deltas. These functions manipulate the network data structure (fig. 2)

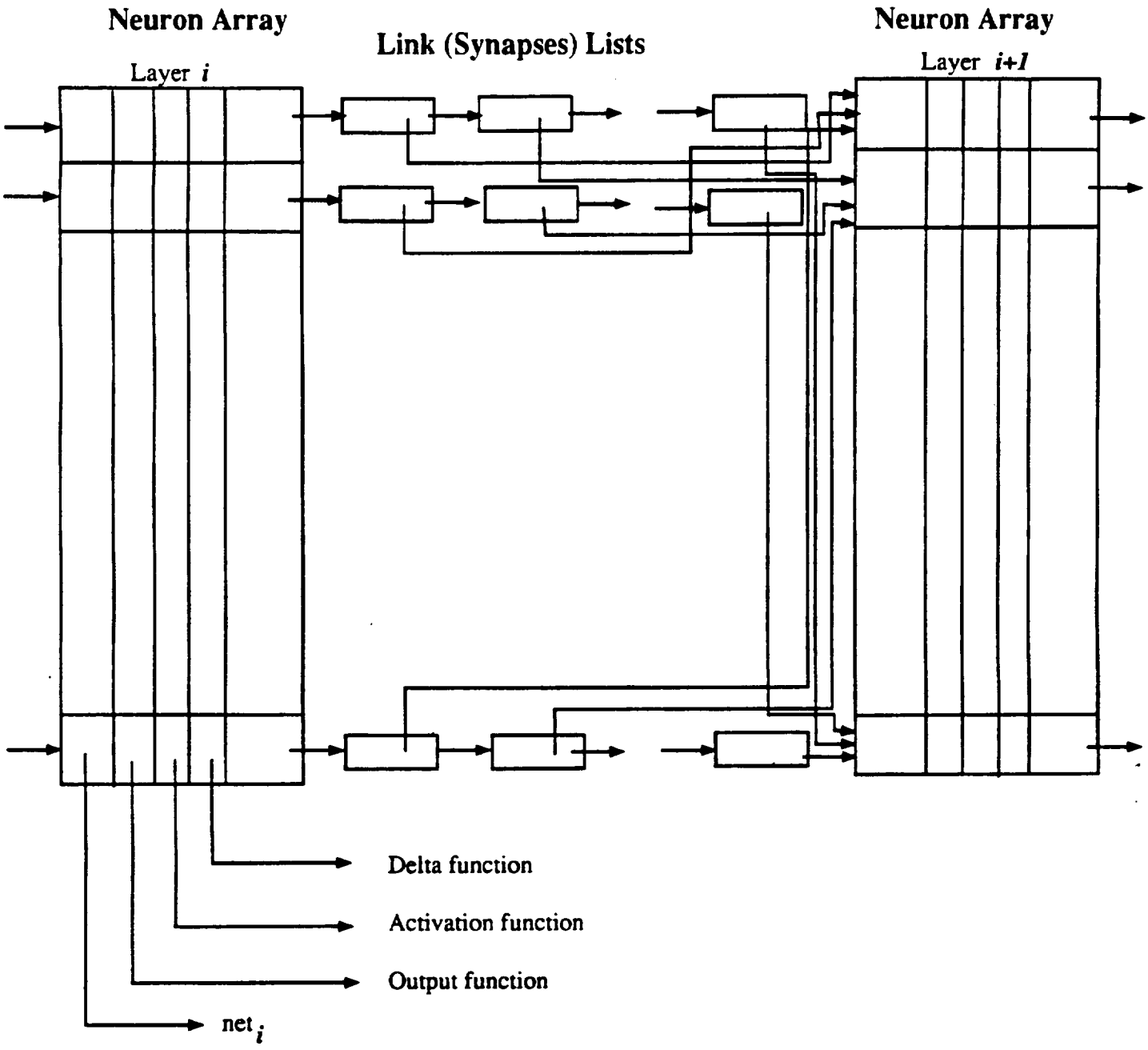

fig. 2 : A sample Data structure for storing a Neural Network in N-CLIPS

for performing the above mentioned functions. This neuron is specifically suited for representing the hidden layers of a feed forward neural network. The rules for input and output layer neurons are slightly different. These special rules can be modified via functions provided in the system to represent any other kind of neural network model. A more generalized model of a neuron is in design.

6.1.2 Artificial Synapse

These are the links between neurons, and are mainly characterized by the following parameters: 'from' and 'to' neuron # and layer #, the type (in or out link), weight. They are stored in a special data structure (see fig. 2) and can also be stored as facts; as in the case of the outgoing links from the output layer neurons. They can be created/edited and deleted as individual links or as a group (from one layer to another). Individual links can be created as C functions or from within CLIPS interpreter (a template possibly from within a windowing system) and group links can be created through a X windows graphics link map editor (explained later). This way fractional (percentage of total neurons) connectivity between layers can be represented very easily.

6.1.3 Activation functions

A library of different existing activation functions is provided to which a user can add a function or modify or delete a function. These functions can be selectively applied to individual neurons or to a group of neurons.

6.1.4 Input/Output functions

Different input/output functions, for neurons in the input/output layers, which are currently popular are provided in a library. The user can add, modify or delete a function from the library. The user can select a function from this library to apply to a single neuron or to a group of them. The input function is usually a linear function, nevertheless a different input function can also be provided. Also for single layer feature maps [10] the input functions could be much more complex. In N-CLIPS this complexity can as well be mapped directly in a neuron rule.

6.1.5 Threshold types

A high pass threshold is the most general type used, where if a neuron's activation is above a certain threshold it fires. A low pass threshold type is characterized by its ability to allow a neuron to fire only if its activation is below a certain threshold. The band pass (and the multiple band pass) threshold types [12] are applied when a neuron fires if its activation is within a single range of values or several ranges. These are available as choices when the user is describing a neuron and can be applied to a solitary neuron or a collection of them.

6.1.6 Constants of the Equations

The constants applied in the various equations can be changed during the network training sessions via the user interface provided by the system. Momentum factor, and Learning rates are two such constants which are applicable to the back propagation neural networks. Different momentum factors and learning rates can be applied to different parts of the network.

6.1.7 Delta functions

Delta functions, as prescribed in [3], are available in this system. Users can also add customized

delta functions to the library.

6.1.8 Error Criteria

While the mean squared error is the most generally used error function, and is the one currently supported, future extensions will provide for other error criteria (e.g. entropy).

6.2 OBJECTS (Composite)

6.2.1 Layers

This system provides both layered and non-layered neural networks. Neural layering allows for grouping of neurons wherein information is passed between a group of (layer) and its two 'nearest neighbours (layers)'. Information flow between neurons of the same layer (horizontal connectivity) is also permitted. The layers can be created, edited or deleted by the user through the system provided functions. The parameters are accepted via a template provided to the user, after which the parameters are encoded and saved in the network data structure (fig. 2).

6.2.2 BPNN

A multi-layer feed forward neural network which follows the generalized delta learning rule is provided with modifiable parameters. The user can specify in the BPNN template the number neurons/layer, the number of hidden layers, the bias (threshold) values, the input/output and activation function, layer specific learning rates and momentum factors and other parameters from a list default and optional parameters provided by the system. The user can also update the links between neurons by the link map editor.

6.3 ACTIONS (Primitive)

6.3.1 Create, Edit & Delete Neurons, Synapses

The user shall be given a library of functions for creating and modifying the above mentioned objects. The create_neuron function can be called from within a C program or from the CLIPS interpreter just like defrule. In CLIPS> the user can enter the parameters of a neuron from the template provided. The template will carry default parameters and also provide help on different options available for each parameter. The parameters have to be passed to the create_neuron function if called within a C program. The function will encode the parameters into a special rule and shall also update the network data structure (fig. 2). The function for creating a synapse is called create_synapse and it also is C and CLIPS callable. The synapse information though is only stored in the network data structure. Other functions like edit_neuron and edit_synapse, are basically invoked in the CLIPS interpreter. They let the user modify the values of the neuron/synapse parameters. The delete_neuron functions simply take the neuron and layer numbers and delete the neurons and the links from/to them. The delete_synapse requires the 'from' neuron and layer numbers and the 'to' neuron and layer numbers. The network data structures and clips data structures are updated accordingly.

6.4 ACTIONS (Composite)

6.4.1 Create, Edit & Delete Neurons, Synapses

When a group of neurons or synapses have similar characteristics they can be created, edited and deleted by a single function call. Functions to create, delete and edit a group of neurons and

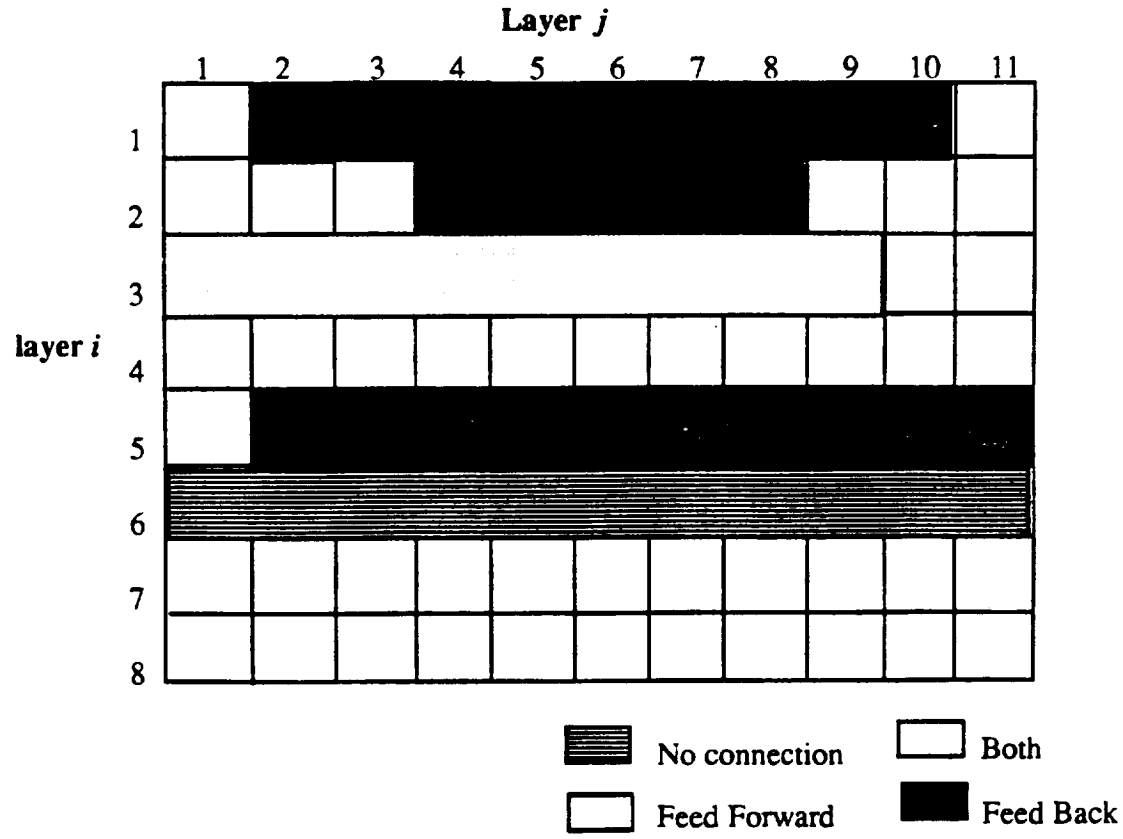

fig 3a. X Windows Link Map Editor : Modifying links, an example.

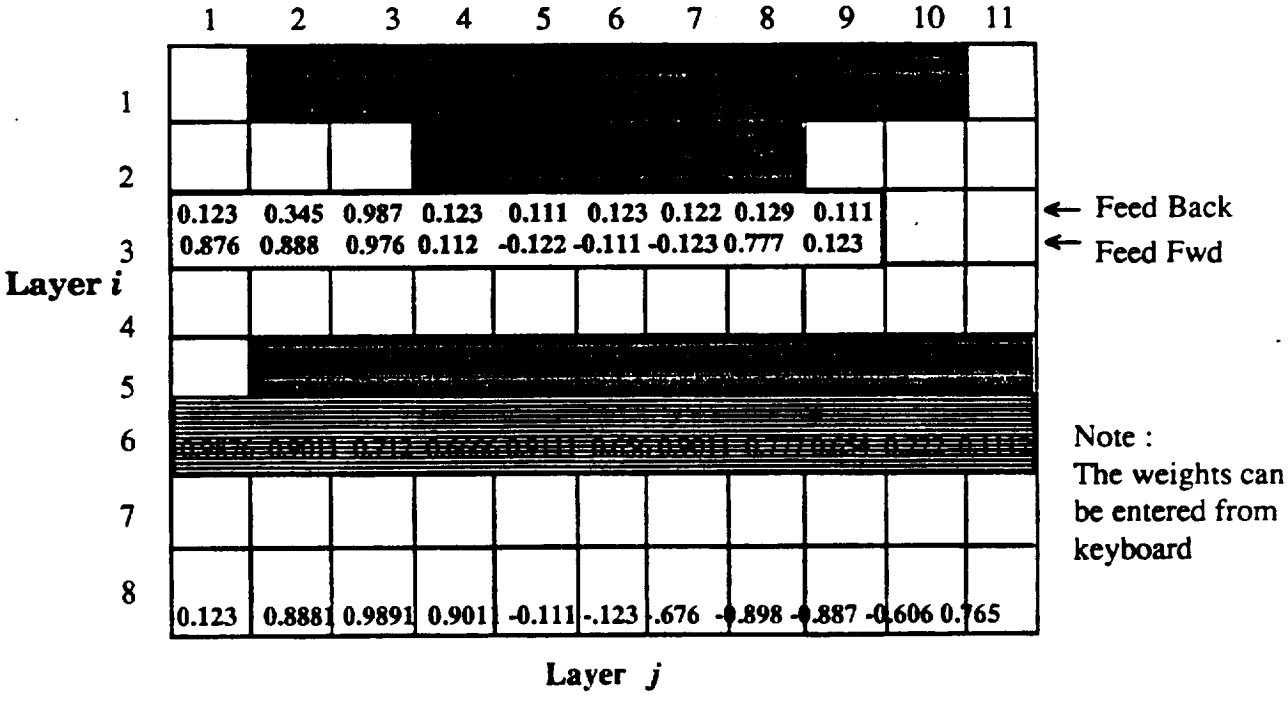

fig 3b. X Windows Weight Editor : Modifying weights, an example.

synapses are provided in the function library. As in the case of primitives, these functions (for the actions) can also be accessed both, from within a C program and from the CLIPS interpreter. The template invoked from the interpreter, however would request additional information from the user apropos the number of neurons, synapses or the layers under consideration, their topological relationship etc. The group is treated as a composite object in the system which stores it as a collection of possibly inter-connected primitive neurons and synapses. These groups can be connected to other groups, though it is a very difficult task to determine the actual neuron to neuron connection as it could be a one-to-one, one-to-many or a many-to-many from one group to another. Also, the connection from one group to another can be a higher level, logical (or abstract) connection. Besides these there can be a neighborhood effect [10] which can be programmed into the group as a rule. The creation and editing of groups of synapses is carried out with an X-Windows link map editor explained next (fig. 3a). The weights of the links can be changed through a similar graphical editor.

6.4. 2 X-Windows link map editor

It is a two dimensional link map where the rows represent the 'from' neurons on a layer and the columns represent the 'to' neurons in another (defaulted to next). It has a mouse interface to switch between four types of connections, namely the feed-forward (black color), feed-back(white color), none at all.(B&W pattern 1), both (B & W pattern 2). After the user has created or modified the links between two layers and has saved them, the map will return a matrix with the values (-1,0,1,2) for feed-back, 'none', feed-forward or 'both' connections between neurons. The user could then either use that matrix to create his/her own link specs in a C program or can let the library function create and modify the data structures. The map has default link connection specifications to create the links automatically.

6.4.3 X-Windows weight editor

It is the same as the link map editor in appearance and functionality with the exception that the user can enter the weights or modify them manually for each type of synapse at the time of creation or at any point during training, even during the execution (fig. 3b).

6.4.4 Create, Edit & Delete Layers

These can be created via direct function calls to create layers, or can be built incrementally by first creating the other sub-components of the layers. The layers can be of basically three types input, output and hidden, though feature maps usually have only one layer. The system provides functions to create a standard layer or a group of them. These can be edited as individual layers or a group of (hidden) layers. Once all the neurons on a layer are deleted, the layer automatically collapses. Deleting a layer would result in all connecting synapses being purged too. If a hidden layer is deleted resulting in partition of the network the user shall be prompted with available options which would include destruction of the network and default connections.

6.4.5 Create, Edit & Delete Networks

A user can create, modify and even delete complete neural networks. In this system the user will have the capability of creating his/her own networks by either modifying the system defined neural networks (BPNN, currently, is the only available Neural network) or by customizing one of his/her own.

6.4.6 Ntrain

This function is a set of expert system rules (in CLIPS) which is system defined for feed forward type networks. But the user can write his own training function, if desired. The system defined training function first reads the input pattern and then systematically triggers each layer. To write ones own training function the user will have to write an expert sub system which will then override the previously defined training function. It could be possible to have different training functions if the network consists of different learning algorithms as sub networks. Since there can be more than one network active at any given time, the training functions should be classified by the network number to which they pertain.

6.4.7 Nrun

The neural networks or sub networks can be run from a CLIPS interpreter, a C program, or can be spawned off from CLIPS rules. Since there can be more than one network active at any given time, hence this function also needs to be passed a network identifying number.

6.4.8 Freeze

This function pauses the execution of the network after which the save function can be called to save the snap shot of the system for later analysis.

6.4.9 Show_ready

If the user wants to know, at any given instance, which set of neurons is ready to fire, he can invoke the show_ready function. This function provides a display, either in the form of a list of neurons or as a change of neuron color in a graphical representation of the neural network interconnections. The function can be invoked via a mouse.

6.4.10 Save, Load

A neural network can be saved at any given time in the disk files via the save_nn() and load_nn() functions. The save function saves all the rules in appropriate files and also the data structure associated with that network. The load function reads the same files and builds the neural network representation within the system.

7. CONCLUSIONS

N-CLIPS has turned out to be a very useful tool for solving real life technical problems for which a single knowledge representation or AI technique does not suffice. The building-in of a neural network simulator within CLIPS (the expert system shell) made it easy for the two to communicate with each other, share a common fact (data) base and utilize the other's strengths to overcome its weaknesses (e.g. expert systems brittleness versus the neural networks associative capabilities). The problem of mapping one system into another is a very difficult research topic to be addressed in future extensions of N-CLIPS. As far as the neural network paradigms are concerned, we plan to add all known learning paradigms as stand alone objects. The user-interface, can be enhanced to a complete windowing environment (e.g pop-up menus, mouse selectable options list, graphic templates, etc). The most important enhancement to the system would be the incorporating of semantic networks, searching algorithms, more general connectionist networks, frame based systems, and even petri nets.

8. REFERENCES

[1] Hoskins J. and Jones W. "Back Propagation", BYTE , Oct 87.

[2] D'Autrechy C.L., Reggia, J.A. "MIRRORS/II, Connectionist Simulation", First Annual INNS Meeting, Boston, 1988.

[3] Rumelhart et al, "Parallel Distributed Processing", vol I., 1987, MIT Press,Cambridge,Mass.

[4] CLIPS User's Guide, Artficial Intelligence Section, Johnson Space Center,June 1989.

[5] McGee B., Miller M., Krolak P., and Barr S., "Interfacing PCLIPS into the Factory of the Future", "The First CLIPS Users Group Conference", NASA J.S.C., Houston, Texas, Aug. 1990.

[6] Rabelo L.C. and Alpteking S., "Synergy of Neural Networks and Expert Systems", Proceedings of the Third TIMS/ORSA Conference on FMS, MIT, Cambridge, MA., Aug. 1989.

[7] Bhatnagar H., Krolak P., and McGee B. "A Traffic Controller for Material Handling Systems", submitted to SOAR Conference, Albuquerque, New Mexico. June 26-28, 1990.

[8] Miller R., Korlak P. "PCLIPS : A Distributed Expert System". "The First CLIPS Users Group Conference",NASA J.S.C., Houston, Texas, Aug. 1990.

[9] Coleman J. "Evolutionary Telerobotics: An Approach to the Designing of Telerobotics System", #CPE-NERV-90-5, Center for Productivity Enhancement, University of Lowell, Lowell, Ma. 01854.

[10] Kohenen T. "Self Organizing and Associative Memory. " Springer-Verlag, New York. 1989.

[11] WIlliams R. J. "Inverting Connectionist network mapping by back prop error." Proc. 8th Ann. Conf. Cog. Sci. Soc. 1986.

[12] Gelband P. "Neural Selective Processing and Learning, " Proc. of the First Ann. INNS Meeting, Boston, 1988.

[13] Collins E., Ghosh S. and Scofield C. "An application of a Multiple Neural Network Learning System to Emulation of Mortgage Underwriting Judgements, " Nestor Inc., 1 Richmond Sq, Providence RI 02906.

[14] Reilly D., Scofield C., Elbaum C., and Cooper L.N. "Learning System Architectures composed of Multiple Learning Modules".

[15] Reily D. et al., "An application of a Multiple Neural Network Learning System to Industrial Part Inspection," ISA, 1988, Houston, Texas.

[16] Pineda F.J., "Generalization of Back-Propagation to Recurrent Neural Networks, " Physical Review Letters, Nov. 1987, pp 2229-2232.

[17] Hopfield J., and Tank D.W. "Computing with Neural Circuits," Science 233, 625-633 (1986).

8. REFERENCES *Contd.*

[18] Kirpatrick S., Gelatt C.D., and Vecchi M.P., "Optimization by Simulated Annealing," Science 220, 671-680 (1983).

[19] Grossberg S., Carpenter G.A. "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," Chapter 5., Neural Networks and Natural Intelligence, MIT Press, Cambridghe, Massachusetts,1988.

[20] Kosko B. "Bi-Directional Associative Memories, ". IEEE Trans. on systems, Man & Cybernetics, vol 18, pp 49-60, 1988.

APPENDIX

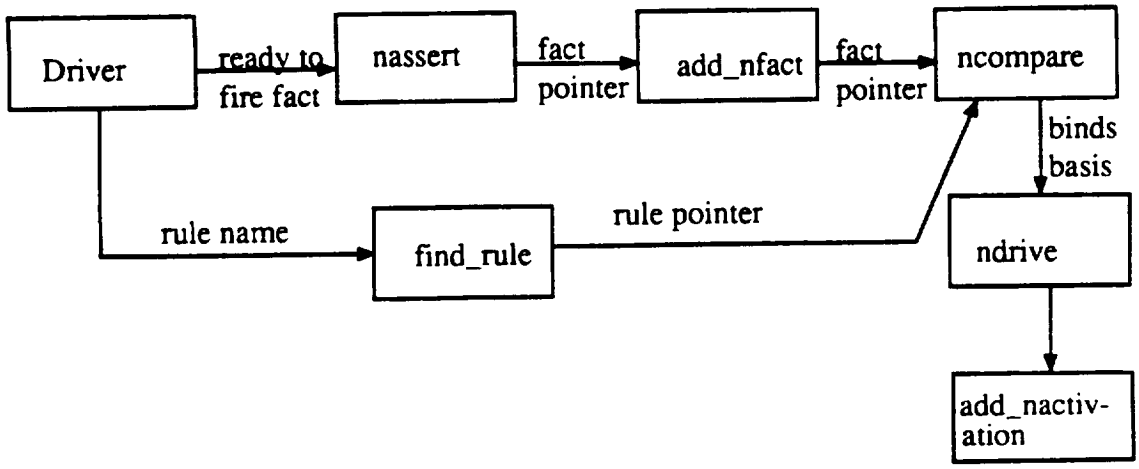

fig. 4 : **A data flow diagram of the changes made to CLIPS for N-CLIPS**

Driver

This function goes through an array of neurons (a layer) and for each neuron that is ready to fire it calls find_rule to set up a global variable pointer which points to the current neuron rule. This is followed by a call to nassert to assert the following fact : (neuron # layer # ready to fire).

Nassert

It calls add_nfact() with the above fact after making sure it has not been asserted already.

Add_nfact

It adds the above fact to the fact list and calls ncompare to filter through the special neuron rule.

Ncompare

It make s the var list (binds), the joins and gets the rule pointer from the global variable and then calls ndrive to drive the fact through the network patterns for that rule .

Ndrive

its task is to put the input parameters in proper data structures and calls add_nactivation to add the rule to the agenda.

An important feature of the above functions has been that only one rule and one fact is in picture. this is done since we know both the fact and the rule which its assertion will trigger. However in case of output neurons other facts are asserted which could trigger an expert system.

Nretract

It retracts the ready to fire fact from the fact list after the neuron has fired.