TECHNICAL RESEARCH REPORT

Institute for Systems Research

Linking Symbolic and Subsymbolic Computing

by A. Wilson and J. Hendler

The Institute for Systems Research is supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

TR 93-12

Linking Symbolic and Subsymbolic Computing

A. Wilson J. Hendler*

Department of Computer Science
University of Maryland
College Park, MD 20742
wilson@cs.umd.edu
hendler@cs.umd.edu

Abstract

The growing interest in integrating symbolic and subsymbolic computing techniques is manifested by the increasing number of hybrid systems that employ both methods of processing. This paper presents an analysis of some of these systems with respect to their symbolic/subsymbolic interactions. Then, a general purpose mechanism for linking symbolic and subsymbolic computing is introduced. Through the use of programming abstractions, an intermediary agent called a *supervisor* is created and bound to each subsymbolic network. The role of a supervisor is to monitor and control the network behavior and interpret its output. Details of the subsymbolic computation are hidden behind a higher level interface, enabling symbolic and subsymbolic components to interact at corresponding conceptual levels. Module level parallelism is achieved because subsymbolic modules execute independently. Methods for construction of hierarchical systems of subsymbolic modules are also provided.

*This research was supported in part by a fellowship from the American Association of University Women to A. Wilson, and by grants from NSF(IRI-8907890), ONR (N00014-J-91-1451), and AFOSR (F49620-93-1-0065). Dr. Hendler is also affiliated with the UM Institute for Systems Research, an NSF-sponsored Engineering Research Center.

1 Introduction

In recent years, the disparity between connectionist and traditional AI systems has been reduced by an increasing number of systems which integrate symbolic and subsymbolic computation. For some, this integration is motivated in part by the observation that human intelligence seems to allow symbolic processes such as language and long-term planning to be supported on the neural network of the human brain. For others, it is motivated by the engineering need to develop applications such as intelligent controllers and signal understanding. Whichever motivation, an analysis of existing systems shows that most hybrid systems have been tailored to specific applications, or have focused on improving the behavior of systems in particular domains such as planning, rule following behavior, and natural language processing.

In contrast to such specialized system development, in this paper we describe work which focuses on symbolic/subsymbolic interaction per se. Our goal is to develop some general purpose methods for integrating symbolic and subsymbolic components, and for supporting the computational needs of hybrid systems developers. As such, a critical aspect of our work is understanding the relationship between the symbolic and subsymbolic levels of hybrid systems in order to support the many different ways in which these systems are integrated.

To understand the relationship between the levels in hybrid systems, we have studied how a number of current systems allow for symbolic/subsymbolic interactions. For each system, the first characteristic we assessed was the degree to which symbolic and subsymbolic processing were interwoven. In some cases, the two subsystems are very highly integrated, to the point where it is difficult to designate a boundary between them. For example, in systems which propagate both symbolic markers and activation values over the same data structures (cf. [LHFB89, KTL89]) it cannot easily be said whether the systems are symbolic or subsymbolic for any given computation.

A second class of systems provides only very loose coupling between the symbolic and subsymbolic components. In these systems, the computations are separate, and any information flow between them is carefully regulated. For example, in the SCRuFFy system of [HD91], a backpropagation network functions separately from an expert system, and a special purpose integrating mechanism is used to provide output from the network level to the expert reasoner. Although useful in practice, such a system is very limited in terms

of the interactions provided between the levels.

Current research, however, tends to use more general methods than the latter, but without the complexity of integration of the former. That is, although these system provide for a tight coupling between the symbolic and subsymbolic reasoners, the program control tends to flow from one method to the other, so at any point the processing can be characterized as symbolic or subsymbolic. In contrast to the continuous, fused interaction of the highly integrated systems, which is difficult to capture in a general way, we found that a number of these systems had important commonalities that could be identified as targets in the development of support tools.

Table 1 summarizes a number of these systems, and illustrates some similarities as well as differences in interaction methods. Perhaps the most important point of commonality is that the "goal" of each system tends to be symbolic in nature. Accordingly, the inputs to these systems are typically symbolic, and subsymbolic processing is used as an important adjunct in computing a some sort of a symbolic representation.

In the systems shown in Table 1, connectionist networks may be created a priori (before the program is invoked), once per program invocation, or once for each input pattern received by the program. Some of these systems use networks which are created manually [Hen91, Sun91, WL89], while the others create networks automatically [Lan91, Leh91, TS92].

We found that each of the natural language processing systems uses dynamic network creation or selection based on the current input. Two of the these systems create a unique subsymbolic network for each input sentence or phrase [Leh91, Lan91]. The third [WL89] uses a different dynamic methodology. In this system, a suite of networks is trained before the program is invoked. Then, for each input a subset of these trained networks is selected for use. In contrast, each of the other three systems uses one fixed subsymbolic network per invocation. These systems were developed for improving the traditionally symbolic methods used in developing rules [TS92] and performing common sense reasoning [Sun91] and planning [Hen91].

The goal of the particular system provides some insight into whether the subsymbolic component is created dynamically or statically. In those cases using static information about a domain, networks are fixed. In contrast, in natural language processing, the static information available to the system pertains to general syntactic or lexical information. Yet an understanding of the semantics of an input is often required to properly parse it. Obviously,

	Hendler 1991	SAARCS Lange 1991	CIRCUS Lehnert 1991	CONSYDERR Sun 1991	Towell, Shavlik 1992	Wermter, Lehnert 1989
System Goal	Improve semantic net performance in a planning environment	Sentence understanding, analog retrieval	Sentence analysis	Avoiding brittleness in commonsense reasoning	Extracting rules from a neural network	Noun phrase understanding
System Input	symbolic goal -> activation pattern over localist nodes	sentence -> activation pattern over localist nodes	Sentence	query -> activation pattern over localist nodes	Symbolic domain info	Noun phrase
System Output	activation pattern over localist nodes	activation pattern over localist nodes	Completed Frame	activation pattern over localist nodes	Symbolic rules	Structural interpretation
Network Creation	a priori	Once per input	Once per input	Once per invocation	Once per invocation	a priori
Symbolic to Subsymbolic Influence	Output layer size dependent on number of a subset of semantic nodes	Topology dependent on input	Topology dependent on input	Topology dependent on structure of localist net	Topology dependent on symbolic domain info	Topology dependent on encoding of preselected microfeatures
Input to Subsymbolic Network	Activation spreads from semantic network	Activation spreads from semantic network	Activation levels chosen based on semantic info	Activation spreads from localist network	Subsymbolic network used only in training	Manually determined encoding of preselected microfeatures
Meaning of Subsymbolic Output	Similarities over perceptual encodings	Most coherent interpretation	Most likely site for prep. phrase attachment	Strength of presence of microfeatures	Distributed encoding of refined symbolic rules	Plausibility of a particular interpretation
Use of subsymbolic output	Subsymbolic activation used in semantic network activation	Subsymbolic activation used in semantic network activation	Used only when symbolic methods fail	Subsymbolic activation used in localist network activation	Weights and topology used as input to extraction algorithm	Network yielding greatest plausibility has correct interpretation

Note: -> means "translated to"

Table 1: Symbolic/Subsymoblic Interactions in Hybrid Systems

this information is not available until the input arrives.

Three of the systems in our survey use networks which were trained outside the scope of the system. Two use a priori training [Hen91, WL89], while the third trains the network after it has been constructed using a symbolic to subsymbolic mapping algorithm [TS92]. Unlike the other systems, these systems rely on the existence of microfeatures discovered by the network, thus necessitating the training. Note that in these cases separate network training most likely relied on human observation and adjustment. This human influence could be viewed as exerting a symbolic influence on the network.

There is one way in which a symbolic component consistently influences a subsymbolic component. In each system, the symbolic component defines part or all of the subsymbolic network's topology including some initial weight values. This is clearly the case for those natural language processing systems which dynamically create subsymbolic networks. Hybrid systems using parallel symbolic methods often require corresponding nodes in a subsymbolic component [Hen91, Lan91, Sun91]. The last way in which the subsymbolic topology is influenced is as a result of a direct mapping from a symbolic representation to a subsymoblic network [TS92].

Finding such a consistent influence on network topology is not too surprising. It is clearly advantageous to focus the computation of the subsymbolic network as much as possible beforehand. As noted by others, a network's structure plays an important role in its processing by enabling learning and restricting the search space [Die89, Arb87, vdM88].

Influence in the other direction, from the subsymbolic to the symbolic, is generally different in nature. Symbolic components will delegate subtasks to subsymbolic components, but the opposite is not true. Subsymbolic components tend to be used mainly to improve or enhance symbolic techniques. Note that this is consistent with Lehnert's statement [Leh91]:

> "[This analysis] suggests an important claim about the relationship between symbolic and subsymbolic processes which sounds quite plausible in general: Symbolic processes can be influenced by subsymbolic processes, but the converse does not hold."

Of particular note in these hybrid systems is the incorporation of the subsymbolic output. It is at this point that, in a variety of different ways, the subsymbolic output becomes symbolic. In some cases, the output is interpreted discretely and symbolically, such as a plausibility value or an

attachment point. This occurs in those systems in which the main flow of control is serial [Leh91, WL89]. In systems using parallel symbolic processing, the output is simply propagated back to the semantic or localist network [Hen91, Lan91, Sun91]. In these systems, as energy is propagated to the symbolic component it activates some subset of symbolic nodes, thereby becoming symbolic. One system in our survey [TS92] focuses precisely on this subsymbolic to symbolic shift by developing a method for extracting symbolic rules from the subsymbolic network.

Based on an examination of systems such as those discussed above, it becomes clear that a mechanism which can help link symbolic and subsymbolic processing needs to allow for a number of capabilities:

- For those networks requiring it, off line training should be possible at various times during processing. Network tuning should be possible also.
- There should be a way for activation to pass between symbolic and subsymbolic components.
- There should be a means by which subsymbolic output can be translated to a symbolic representation.
- Both static and dynamic network creation should be supported.

In the remainder of this paper we present the *Conncert* system, a framework and set of tools which we have designed for providing these capabilities. We first describe the linking between subsymbolic and symbolic computation by analogy with "abstraction" as used in traditional computing languages, and then show how Conncert supports subsymbolic abstractions. We present some examples of the capabilities that Conncert offers, and describe how it supports the capabilities listed above.

2 Symbols and Abstractions

Our basic contention is that the "symbol" which results from the numeric output of subsymbolic computation is similar to an *abstraction* in the context

of programming languages.[1] Ghezzi [GJ87] defines the process of creating such abstractions as

> "... identifying important qualities or properties of the phenomenon being modeled. Using the abstract model, one is able to concentrate only on the relevant qualities or properties of the phonomenon and to ignore the irrelevant ones."

In programming languages, abstractions provide symbols through the symbolic naming of both locations in memory and operations on those locations. Thus, even an integer is an abstraction, representing both a location in memory and a method for interpreting the bit pattern stored there. Similarly, a function serves as an abstraction for a memory location which can be interpreted as executable code. Put simply, these abstractions raise the level of discourse away from that of the computer and brings it closer to that at which algorithms are described.

In contrast, in the context of AI programming, a symbol often stands for a concept meaningful to a person but which may or may not have meaning to the computer, such as **(son_of clyde)**. Here, symbols are chosen not to aid in directing the computer per se, but because the symbols and their relationships are inherently important to people. As in the case of integers and functions, however, low level details are avoided. These symbols serve as abstractions to allow deliberation at a level similar to that used in daily discourse.

In general, subsymbolic computing does not support deliberation at the same level of abstraction as do the symbols traditionally used in AI programming. In discussing subsymoblic computation, one often talks about weights, activations, input vectors, squashing functions, learning rules, etc. These are all terms describing characteristics well below the level of description of the task performed by the network as a whole, such as pattern completion, categorization, or associative retrieval – the high level emergent behavior is characterized in terms of its low level numerical aspects.

To provide for an integration of symbolic and subsymbolic processing, the results of the subsymbolic analysis must be made available to the sym-

[1]Recent speculations about the existence and role of symbols have examined many different notions of "reducibility" and "irreducibility." Our analogy to programming abstractions can support either view, and thus we leave this matter open to those philosophers so inclined.

bolic system at a level similar to the sorts of symbols used in AI programming. That is, integrating these systems requires providing abstractions for the subsymbolic computation and output to make them more accessible to the symbolic system. Thus, it becomes clear that a model which provides programming level abstractions is necessary to supporting the sorts of capabilities described in the introduction, and a general purpose mechanism must support the creation of these abstractions. (In addition, the benefits usually associated with abstraction, such as increased modularity, information hiding, simplification of code modification and reuse, etc., are made available by such a tool, although we will not discuss this in detail in this paper.)

However, providing abstractions in the realm of connectionist modeling is somewhat more complex than in traditional programming. In traditional models of abstraction such as object oriented programming, operations on objects are implemented via function calls. Generally, the behavior of these operations is easy to understand. Verbal descriptions can be provided which are sufficient to enable them to be used properly by others. Functionality is usually well defined so that a given input always provides the same output, and the computation is relatively simple (algorithmically).

The nature of connectionist computation is very different. Connectionist components are used in cases where it is difficult or impossible to symbolically describe the desired input/output mapping. This mapping can also change over time, as the network learns. Functionally, connectionist networks tend to be general purpose mechanisms, and their specific meaning arises only as they are applied (and usually trained) to a particular problem. Thus, their success relies on proper construction and training within a particular environment.

A programmer may not fully understand network behavior. Indeed, a general method for identifying the features on which a network discriminates is still an open question. Further, depending on the network model and its dynamics, outputs may vary over time so that the same input may produce different outputs depending on its position in the input stream. Due to this dynamic nature, outputs may or may not reflect the desired mapping – quite different than in the case of traditional functions.

Another difference between subsymbolic computing and more traditional programming is that the answers produced by connectionist methods are typically distributed over a set of nodes and must be mapped to a symbolic interpretation. It is most often humans who are employed to perform this

translation. For example, a network performing categorization may make one output level high and the rest low, but it is people who interpret that output as symbolizing a particular category. In addition, outputs tend to approach their limits rather than actually reach them, and thus the human may also have to decide what criteria are used to determine high vs. low outputs.

There are further complications involved in connectionist model usage. Most models involve some degree of convergence to produce the desired mapping or to reach a state of equilibrium. Usually, a person is involved in making the determination of whether the convergence has occurred or will ever occur.

Finally, even through connectionist models are cited for their robustness in the face of noise or novel inputs, they often can not respond to a changing environment. Instead, if it is possible for an environmental change to occur, a person is employed to monitor the output of the network to determine whether retraining is necessary. If so, the network may be taken off line, retrained, and placed back on line, all by hand.

Humans play an important role in analyzing, interpreting, and adjusting connectionist models. Lange (p. 34) refers to the problems requiring human intervention as *unreduced mechanisms.* Currently, researchers are investigating methods for dealing with these issues, but much work remains to be done.

These differences between connectionist computing and traditional programming (both in AI and elsewhere) provide an obstacle to creating general purpose abstractions for connectionist computing, and thus impede the development of hybrid models. To overcome these difficulties, a framework is needed which provides a method for encapsulating connectionist systems, and removing the need for human interaction at the boundary between symbolic and subsymbolic computing.

3 Network Encapsulation Through Supervision

To provide for the integration of subsymbolic computing into more general symbolic systems[2], we have developed an intermediary mechanism that encapsulates the subsymbolic computing. This allows connectionist models to truly become "functions" in the traditional sense, and thus allows for abstraction as used in other programming methodologies.

We call our symbolic/subsymbolic intermediary a *supervisor*, or a *super*. A supervisor is a software agent which provides an interface to and monitors the performance of a connectionist model. Supervisors and their respective networks can be created using a system we have developed called Conncert. Conncert consists of a set of objects for creating networks and their supervisors, written in the object oriented language C++.

In Conncert, a supervisor is associated with each subsymbolic network. The supervisor provides a symbolic interface to the network through which high level information can be passed. At the same time, the boundaries between components are maintained, reflecting their logical distinctness. Our goal in developing an intermediary is to be able to create a fully encapsulated **connectionist module** having a high level, symbolic interface, usable like any other software module.

The supervisor fills three important roles. First, it controls the environment in which the network functions. Environmental control duties vary across models, and may include controlling the spread of activation, making transitions between phases (such as training versus recall phases), and adjusting parameters.

The second job of the supervisor is to present the network's answer. This means that the supervisor must determine whether or not a legitimate answer is present. This raises several issues. In some cases, the supervisor must determine whether the network has converged. If not, the super must decide whether it will ever converge and possibly take some corrective steps such as adjusting parameters, modifying the topology, or simply halting and reporting the problem.

Even if the network has converged, the super must further determine the legitimacy of the output. It may be the case that the output of the network

[2]And, for that matter, other computational environments

is erroneous or meaningless, in which case the super must respond or indicate this in some way. If the output is legitimate, the super may perform the task of translating from a subsymbolic output vector to a symbol.

The third role of the supervisory is to encapsulate and provide an interface to the network. All requests to the network and all output from the network must pass through the super. In this way, a distinct boundary is maintained between the network and the rest of the system. Also low level details of network operation are hidden.

Figure 1 illustrates an encapsulated back propagation network module. Inside the module are variables called *environment variables* which are the parameters used in the computation. Also, note that there are two seperate paths for inputs to the module, one each for data and control. Control inputs consist of high level requests to the network in the form of function calls. We call the connectionist modules created in Conncert *network objects* as they behave like any other object in an object oriented programming environment.

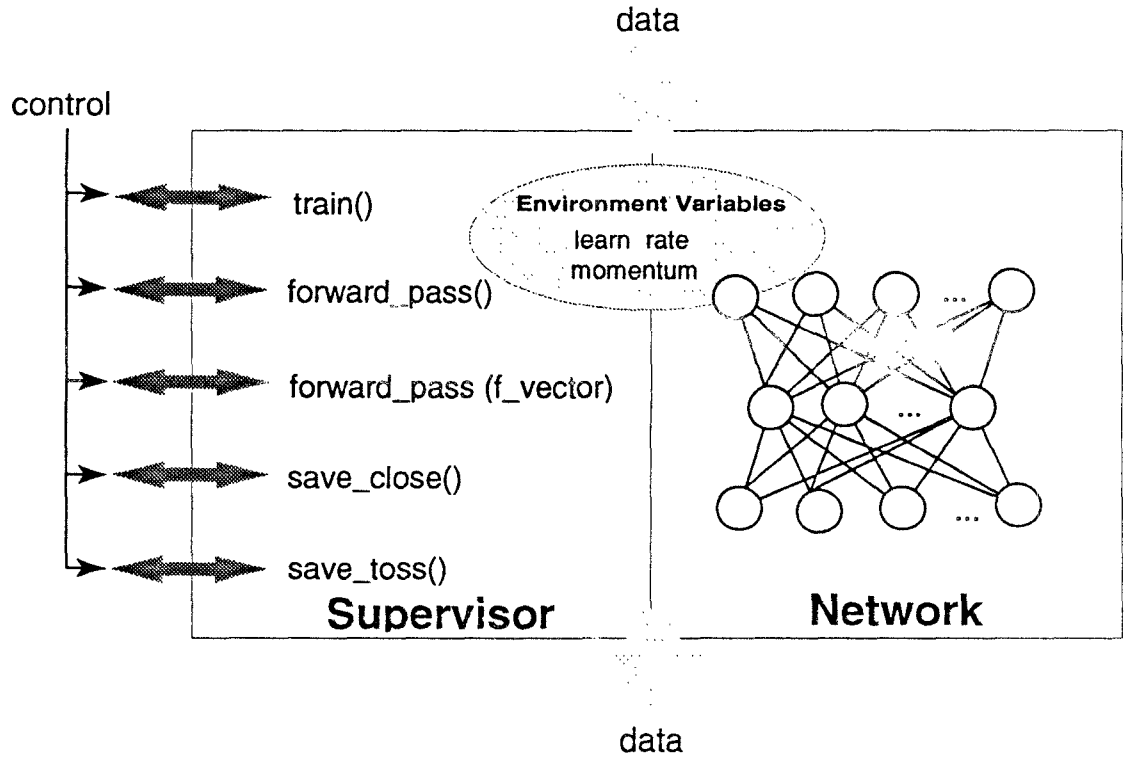

Figure 1: Back Propagation Network Object

3.1 Handling Model Dependent Variations

The various subsymbolic computing approaches have widely disparate natures and monitoring needs. For example, training a back propagation network to perform a particular mapping is very different than tuning a Hopfield network to solve an optimization problem. The former requires multiple synchronous passes of data through a feed forward network while analyzing the output to see that the error is being reduced. The latter requires selecting a set of parameters then observing the asynchronous operation of the network to see if the activation values are reaching a stable state. Diffferences such as these make it not only difficult but imprudent to attempt to build a single supervisory mechanism for all connectionist models.

For this reason, a supervisor is not a mechanism for monitoring a connec-

tionist model per se. Rather, it is a *framework* for monitoring a connectionist model. The difficulty of meeting disparate monitoring needs is overcome by what we refer to as *user tailored programming* and illustrates why the supervisor is a framework and not a single, complete mechanism for all connectionist models.

In user tailored programming the system creates templates in the form of function definitions which are complete except for their contents. A simple example is an environment for building graphical interfaces. Here, the programmer may select an interface object graphically, such as a button or a menu. The system then creates the text for the object's code, but leaves the functionality to be defined by the programmer. The programmer can then add code particular to the application at hand.

In Conncert, in addition to the system functions provided, we also provide certain *template functions*, functions whose contents are intended to be augmented or replaced with algorithms particular to their networks. In particular, the main control loop of the supervisor is a template function. The control loop is a large switch (or case) statement. Referring back to Figure 1, each possible control signal has a corresponding switch in the control loop. The programmer is free to add new switches and add or substitute new functions for the switch bodies. These model dependent control tasks may be implemented in any way - the programmer may choose either symbolic or subsymbolic methods to monitor the network. For example, one could imagine using another subsymbolic network to adjust tuning parameters like learning rate or momentum.

3.2 An Example

To see more directly how Conncert's mechanism for abstraction allows the construction of a hybrid system, we describe the implementation of one such system [Hen91] using Conncert network objects. The hybrid model consists of a semantic network integrated with a three layer distributed network. The distributed network is trained beforehand to classify instances of concepts which appear in the semantic network. As a result, the hidden layer of the network develops its own internal representation using microfeatures that it discovered during training. The distributed network is then attached to the semantic network, by creating edges between the output layer of the distributed network and the corresponding instance nodes in the semantic

network. (The input layer is no longer used.)

In using the system, markers are propagated through the semantic network. When these markers reach those instance nodes to which the subsymbolic network is attached, activation values corresponding to the energy of the symbolic markings are passed to the distributed network. These values propagate down from the output layer of the network to the hidden layer using a specialized activation spreading method. Activations are then propagated back up and out to the semantic net. In the semantic net the instances which were most similar, as determined by the distributed network, gain energy and are able to propagate symbolic information. In this way, paths through the semantic network are established.

Figure 2 depicts this system using a network object defined in Conncert. Here, rather than having actual edges between the two networks, activation values are grouped into a vector, which is presented to the network object. The supervisor of the network object propagates the input vector in the requisite specialized manner. It then groups the output of the network into a vector and passes it back to the semantic network.

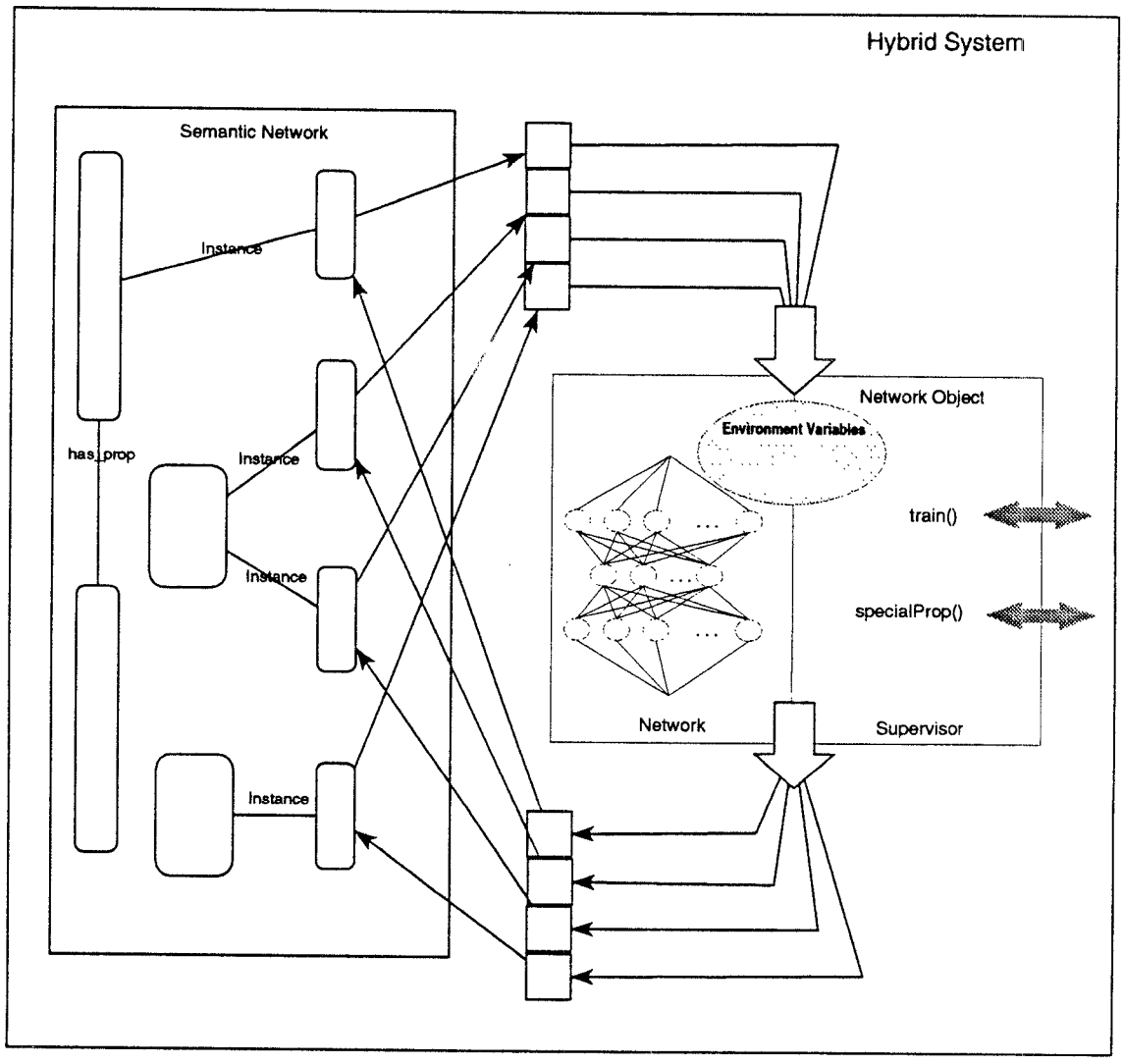

Figure 2: Hendler's Hybrid System Using a Network Object

To create this particular network object, it is necessary to write a specialized activation spreading method for the supervisor. This means simply writing a new method to propagate activation from the output layer to the hidden layer then back to the output layer. Also, a modified function for calculating the net input to nodes in the hidden layer is required to encode the specialized backward flow of activation. These modifications are simple

because high level operations are provided for all objects in the system, including nodes, edges, and sets of nodes or edges (such as a layer of nodes). Existing back propagation operations can be used for other operations such as training.

Moreover, in using a Conncert network object, if desired, the system can be trained at the time the program is invoked. Currently we use a very simple method to determine if the network has converged. For the last N input/target pairs presented to the network, where N is some integer determined to give a useful window size, if the error for each pair was smaller than some epsilon value, we consider the network trained.

4 The Network Object Interface

The interface to a network object consists of a set of functions selected by the network designer to provide high level network operations. There are two essential components of each network object interface function.

In many network computations it is the case that some decision must be made as to the trustworthiness of the network's output. In Conncert, it is the job of the supervisor to make this assessment. Upon every network request, the value of this assessment is reported back in the form of a *confidence value* in the range of zero to one, with zero reflecting no confidence. Systems having network object components can use this value in subsequent decision making. In some situations a confidence is clearly zero, as in a case where the network needed to converge but didn't. Usually, however, determing confidence values is difficult, model dependent, and currently open to research. [3]

As discussed earlier, control of connectionist models is also a difficult problem that has yet to be solved. Because of this, the other essential interface component is an error code, which is necessary until the time (if ever) that network control can be fully automated. This is the means by which the supervisor reports back its findings regarding the success or failure of the network's computation. For example, it is through the error code that the supervisor can report that the network failed to converge. Like the confidence value, the error code is intended to be used by the subsuming system, but

[3] Pertainent research on providing confidence values has been done by Smeija and Muhlenbein. This work is mentioned in the Related Work section of this paper.

rather than reflecting the quality of the output it reflects conclusions drawn by the supervisor regarding the success of the computation.

Thus, the interface to a network object consists of a set of possible operations in the form of function calls. Each operation returns a confidence value, and an error code. Any additional parameters deemed useful by the network designer may also be included. Such an interface is compatible in most programming environments, including most traditional symbol processing systems.

5 Module Level Parallelism and Hierarchical Scaling

We have stressed encapsulation as one of the goals of our system. This encapsulation gives us the usual benefits associated with modularity, but it also provides further benefits specific to this context. Because module interdependencies are minimized, restricted to information that can be passed in function calls, network objects can execute in parallel. Each network object is implemented as a separately executing process, resulting in module level parallelism. [4]

Interactions between network objects occur via message passing. Messages between network objects consist of high level control requests which invoke functions defined in the interface. At the same time, subsymbolic data flows through the objects using paths which are distinct from those used by control messages. This distinction is illustrated in Figure 1 by the different arrows for data and control.

The objects are designed in a manner which allows their data channels to be interconnected, so that the data output of one object can become the input to another. Conncert also provides various intermediary objects, called *connectors*, which allow data channels to be interconnected in arbitrary configurations.

Additionally, these interconnected objects can be grouped into new objects having their own supervisors. In this way, hierarchical object definitions can be built and the system scales up. We refer to simple network objects

[4] Conncert has been implemented using the Mach operating system, which directly supports multi-threading, multi-tasking, and message passing.

as *basic objects* and objects comprised of network objects or other objects as *composite objects*. The benefits of separate data and control paths can be seen in composite objects. Data flows through connectors in the fashion of a data flow architecture - when all necessary data is available at a site, it is asynchronously processed. The flow of data is controlled within network objects via control signals which they receive.

Figure 3 illustrates a composite object to implement the network of modular competing experts developed by Jacobs *et al.* [JJB90]. Briefly, this system consists of multiple, competing networks whose outputs are gated by the output of a gating network. In learning, the network having the output closest to the target output modifies its weights, adjusting its output closer still to the target. Over time the modules tend to specialize in different areas in the input space.

In Figure 3 two expert network objects and one gating network object are depicted. The other objects in the system are connectors which either direct data vectors to various destinations or process data vectors in simple ways. Solid lines represent data paths and broken lines represent control paths. Near the bottom of the composite object, each expert network output vector is multiplied elementwise with the output vector of the gating network. The resulting vectors are then summed yielding the output of the system. Note that for the sake of clarity, data paths for target data are not shown.

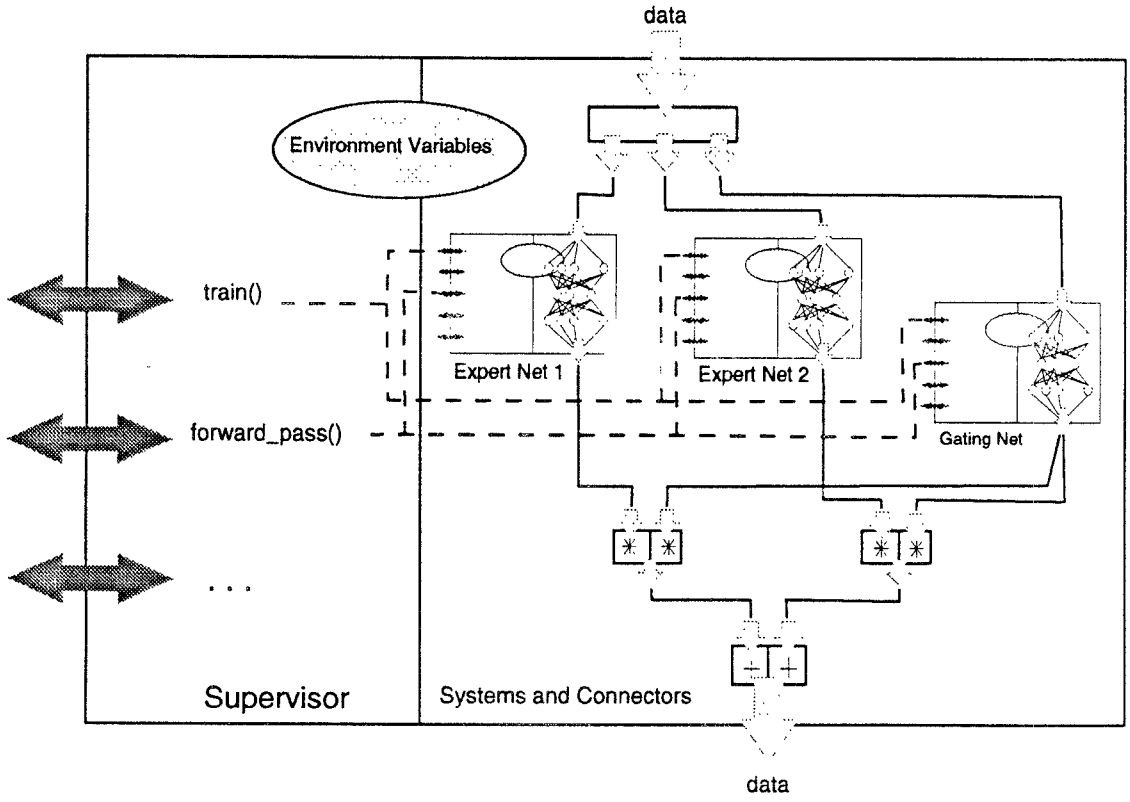

Figure 3: Jacob's Network Composite Object

6 Conncert as hybrid system tool

In the introduction, we described a set of capabilities which were needed to support a number of current hybrid models. Let us now look are how Conncert supports these.

- Off Line Training and Tuning

  For those networks that require it, training is a crucial phase in network construction and use. The ability to incorporate a training phase into subsymbolic network peformance *as it remains a component of a larger system* is one of the strengths of Conncert. The supervisor may halt subsymbolic processing at any time for the purpose of training the

network. It may do this upon receiving an external signal to do so, or, if it is sophisticated enough it may do so on its own initiative.

Network tuning and adjustment is also an important part of subsymbolic network processing. As with training, the supervisor makes it possible to tune the network as it remains a system component. Encapsulating all of the phases of subsymbolic network usage brings us a step closer to being able to build systems that are both fully integrated and fully automatic.

Method for Passing Activations

In our survey we saw that many hybrid systems pass activation values between symbolic networks and subsymbolic networks. In Conncert this is accomplished by grouping activation values into a vector that may be passed into and out of the network object. This method was illustrated in the example using Hendler's hybrid system.

- Method for Translating from Subsymbolic to Symbolic

  We saw that some hybrid systems which use serial symbolic processing have a point in their control flow in which they make a symbolic interpretation of subsymbolic output. Performing this interpretation can be done by the supervisor, so that the output of the network object is a symbol rather than a vector of numeric values. It is by performing this interpretation that we raise the level of the subsymbolic output to correspond to that used by the rest of the system. For example, in Lehnert's system [Leh91], the supervisor would be the obvious choice for translating an activation pattern over the subsymbolic network to an attachment point for the prepositional phrase.

- Static and Dynamic Network Construction

  One weakness of the current implementation of Conncert is that while it supports static network construction, it does not currently support dynamic network construction or modification. However, as we point out in the introduction, we have seen that for some problems, notably those in natural language processing, dynamic network construction may be essential. Extending the system to support this is straightforward, as all the mechanisms for linking nodes and propagating values are in place.

Not only is Conncert capable of handling these types of symbolic/subsymbolic interactions, but at the same time it provides the benefits of abstraction and modularity.

7 Additional Related Work

There are simulators available which provide high level abstractions for connectionist computing [Miy91, GM88]. However, as the goal of most of these simulators is to study connectionist computing per se, most simulators create networks and execute networks in their own environment — not as components of larger systems. While these simulators provide abstractions for the *computation* (such as **feedForward()**, a function to spread activation), they do not provide methods for abstracting the *output* of a network, which is still in the form of a vector of numbers. Abstractions for passing results to other software components are therefore not provided, and it is generally awkward to try to implement hybrid systems using these tools.

Smeija and Muhlenbein [SM92] have addressed the need for subsymbolic network monitoring when networks are used as modules in larger systems. In this work, network monitoring is performed as part of a larger effort to develop a modular neural network architecture. A module in their system, called a MINOS module, is similar to a Conncert network object in that it consists of two components: a worker network and a monitor network. The worker network processes the input and the monitor network provides a confidence value. In particular, this work proposes several different methods for determining a confidence value. In their current implementation, they have chosen a confidence value which reflects whether or not that particular module was trained on that particular input.

However, the goal of this work is very different than that of Conncert. The intent is to develop a new connectionist architecture. Thus, MINOS modules have a fixed interface and are intended to be used as elements in a suite of modules which comprise a larger, subsymbolic system (called a Pandemonium system). As in Jacob's modular system, this system evolves specialized networks subsymbolically. Additionally, this work does not allow for an integration with symbolic processing methods.

8 Conclusion

We have provided a framework for the purpose of making a subsymbolic network an interoperable component of a larger system. Interoperability is obtained by using a software agent to interpret the network output, raising it to a more symbolic level. This way, symbolic and subsymbolic modules may interface at corresponding conceptual levels. Additionally, it is a beginning towards addressing the issues in subsymbolic network monitoring, including training and tuning networks.

Now that a supervisory framework has been developed, much research remains to be done in determining network monitoring methods. General purpose methods are needed, such as determining confidence values and analyzing network performance with respect to convergence. More specific methods must be developed on a per model basis, such as adjusting learning rate and momentum parameters in a back propagation network. Additionally, applying the system to particular problems will entail the development of specialized methods for interpreting subsymbolic output.

Another issue for future work is that, as we mentioned in the introduction, some hybrid systems are highly integrated and it is unclear as to where the boundaries between the symbolic and subsymbolic system are. Conncert, as currently implemented, is only designed to handle hybrid systems where there are clear points of symbolic/subsymbolic interaction. Although we have not yet explored the application of Conncert to such highly integrated systems, it does seem possible to apply Conncert to a network which passes both symbolic and numeric values along edges. In this case it would be necessary to add functionality to pass symbolic values across edges. However, in designing such a system, both symbolic and subsymbolic processing would have to be encapsulated together, which is somewhat contrary to our design intent. Still, there is a potential benefit in having a supervisor bound to such a network. For example, it may extract path information or activation patterns to be used by the rest of the system. We are currently exploring the use of supervision in such systems, with an eye towards extending the framework to support this more complex, but still largely unexplored, level of integration.

References

[Arb87] M. A. Arbib. *Brains, Machines, and Mathematics.* Springer Verlag, 1987. Cited in Diederich, 1989.

[Die89] Joachim Diederich. Instruction and high-level learning in connectionist networks. *Connection Science*, 1(2):161 – 180, 1989.

[GJ87] Carlo Ghezzi and Mahdi Jazayeri. *Programming Language Concepts 2/E.* John Wiley and Sons, 1987.

[GM88] N. Goddard and T. Mintz. *Rochester Connectionist Simulator Manual.* Department of Computer Science, University of Rochester, 1988.

[HD91] J. Hendler and L. Dickens. Integrating neural and expert reasoning: An example. In *Proceedings of AISB-91*, Leeds, UK, April 1991.

[Hen91] James Hendler. Developing hybrid symbolic/connectionist models. In John Barnden, editor, *Advances in Connectionist and Neural Computation Theory*, chapter 7, pages 165 – 179. Ablex, 1991.

[JJB90] Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Technical Report COINS Technical Report 90-27, Department of Computer and Information Science, University of Massachusetts, March 1990.

[KTL89] H. Kitano, H. Tomabechi, and L. Levin. Ambiguity resolution in dmtrans plus. In Manchester University Press, editor, *Proceedings of the Fourth Conference of the European Chapter of the Association of computational Linguistics*, 1989. Cited in Lange, 1991.

[Lan91] Trent E. Lange. Hybrid connectionist models: Temporary bridges over the gap between the symbolic and the subsymbolic. Technical Report UCLA-AI-91-04, Computer Science Department, University of California, Los Angeles, April 1991.

[Leh91] Wendy G. Lehnert. Symbolic/subsymoblic sentence analysis: Exploiting the best of two worlds. In John Barnden, editor, *Advances in Connectionist and Neural Computation Theory*, chapter 6, pages 135 – 164. Ablex, 1991.

[LHFB89] T. Lange, J. B. Hodges, M. Fuenmayor, and L. Belyaev. Descartes: Development environment for simulating hybrid connectionist architectures. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, MI, August 1989. Cited in Lange, 1991.

[Miy91] Yoshiro Miyata. *A User's Guide to PlaNet Version 5.6 A Tool for Constructing, Running, and Looking into a PDP Network.* Computer Science Department, University of Colorado, Boulder, January 1991.

[SM92] F. J. Smieja and H. Muhlenbein. Reflective modular neural network systems. Retrieved from neuroprose electronic archive at archive.cis.ohio-state.edu, March 1992.

[Sun91] Ron Sun. *Integrating Rules and connectionism for Robust Reasoning: A Connectionist Architecture with Dual Representation.* PhD thesis, Brandeis University, July 1991.

[TS92] Geoffrey Towell and Jude W. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In *Advance in Neural Information Processing Systems 4*, pages 977 – 984. Morgan Kaufmann, 1992.

[vdM88] Ch. von der Malsburg. Goal and architecture of neural computers. In R. Eckmiller and Ch. von der Malsburg, editors, *Neural computers.* Springer Verlag, 1988. Cited in Diederich, 1989.

[WL89] Stephan Wermter and Wendy G. Lehnert. A hybrid symbolic/connectionist model for noun phrase understanding. *Connection Science*, 1(3):255 – 272, 1989.