
Questions and Answers about the OWL Neural Network Library

HyperLogic's OWL Neural Network Library is popular among professionals as a way to efficiently implement neural networks in research projects and practical applications. This Tech Note addresses some of the questions of interest to those who want to build neural applications.

What kinds of applications does the OWL support?

The OWL is a completely general-purpose C programming library of neural networks. It is not specific to any particular type of application. The OWL can be employed anywhere a neural network is appropriate, such as financial data analysis, signal processing, adaptive control or pattern recognition.

What functions does OWL have?

All OWL networks are programmed with a common calling interface. What varies from one network to another is the collection of parameters for creating and operating each one. Therefore the number of functions to learn is small, but the interface is flexible enough to handle new network types.

The programming environment provides about 25 functions. These "services" include functions for creating, deleting, training, running, saving, and restoring networks, for reporting the complete network state in printable ASCII form, and for creating error mes-

sages. In addition, there are special functions that address more subtle programming issues such as recovery of specific internal data structures from files.

What documentation is included?

The basic OWL programming manual has well over 300 pages. It includes comprehensive overview information on neural networks, a detailed tutorial illustrating OWL programming techniques, application notes on advanced techniques and data processing, and a complete reference manual.

The reference material includes cross-referenced manual pages for all functions with many usage examples. The network documentation provides the full network specification plus usage notes and programming details.

Can the OWL support real-time and embedded applications?

Neural network code is compute-bound, so sufficient CPU is a necessity. The OWL is a general-purpose C implementation and therefore cannot be as fast as an implementation in optimized assembly language, but it is very efficient overall. The fixed point backpropagation network is particularly good for real-time.

Pre-trained networks are most easily loaded from disk files but you can copy the weights from ROM. With only minor changes to the network creation code, you could use ROM-based weights.

How do OWL neural networks get input data?

The run and train functions take input data in standard C arrays. Your program is free to fill the arrays however you wish. The arrays are usually floating-point, but some networks use integer data. There are no fixed file formats to limit your flexibility.

How many layers does the OWL "backprop" have?

There is no hard limit since the OWL layer data structures are dynamically allocated. Similarly, there are no implementation-imposed limits on the number of nodes in a layer. Thus the only real constraint is the amount of memory available in your system.

Can I see the network weights?

The "brains" of a neural network are its weight arrays. It is vital that you be able to read the weight array for any network. The OWL makes them accessible individually or by layer through a function call. Your program can also access node states and other internal network information.

Can I add my own networks to the OWL library?

The OWL architecture was designed to accommodate user-developed networks. You can add your own networks to the library if you have the source code version. You use the same common network interface to operate your own nets.

Source code documentation tells how to create your own network by writing specific required components. You can use the internal support functions to speed up your development. Or you can start with an existing OWL network — the supplied models can be easily modified into other architectures.

On which systems does the OWL run?

This is really two questions. The first is, "For which machines do you supply a ready-to-use version of the OWL?" Here the answer is given in our current price list.

The second question is, "To

which machines could I easily port the OWL?" The answer is, "virtually any." Customers currently run OWL on VAX under VMS, Ultrix, and Unix, on Sun, Silicon Graphics, HP, and other workstations, on Inmos Transputer and Intel i860 accelerator cards, on many mainframe computers, and a number of other machines and operating systems. Naturally, to port the OWL to such machines requires the source code package.

Backward-compatible PC operating systems such as Windows, Windows NT, and OS/2 support OWL automatically.

What does the OWL require in terms of graphics?

The OWL library itself is completely independent of graphic interfaces. The networks do not interact with the host operating system except for simple file I/O, isolating any graphical interface to the application program.

Most of the sample application programs included with the OWL use a graphical interface to access neural networks. "Native" graphic capabilities are used if possible; IBM versions include Owlgraphics, a simple graphics library included with the OWL for PC. (Owlgraphics comes configured for VGA but is easily modified for other modes.)

What host requirements are there for porting?

The OWL has only minimal interaction with the host system. Mostly this is to perform report and save/restore operations. The "stdio" calls used are easily eliminated by reconfiguration if the target system can't support them.

Other support libraries used are those supporting string functions (strcpy, etc.). These functions are part of almost any reasonable C implementation and pose no difficulties to porting. In most cases it's simply "compile and go."

What "flavor" of C or C++ is it?

The OWL is conditionally compilable with either ANSI or Kernigan & Ritchie syntax. It is written in a highly portable fashion, so almost any standard C compiler is fine. It's been compiled without change for many processors and with many different compilers.

The implementation is C, not C++, but you can use the OWL in a C++ program. With the object code versions for Microsoft and Borland compilers, you need only identify the functions as using C calling conventions. The manual says how to do it.

Does the OWL support Windows programming?

OWL includes a Dynamic Link Library (DLL) for developing applications under Microsoft Windows. The programming interface is identical to the DOS interface — a program handles the networks the same way regardless of which host system is running.

In addition to the DLL, there is a Windows-specific version of the ATOD sample application. The program provides a good working code model of OWL networks running under Windows.

Are there any examples?

Sample application programs are included as code samples that both operate neural networks and employ a graphical user interface. Several currently supplied are listed in Figure 1.

In addition, a portable non-graphical example, "average," illustrates simple command-line

operation. A walk through this program in the manual provides a programmer's tutorial of the OWL.

Can you concatenate OWL networks?

Yes. Because of the general utility and power of network concatenation, this capability was designed into the OWL program model from the start.

For example, the OWL does not supply a self-contained "counterpropagation" network, since the forward version of this paradigm is easily built as the concatenation of two networks that are supplied — the KOH competitive learning layer and the OSL "Outstar" layer. Our CPN example program gives a working code example of how to do concatenation with the OWL.

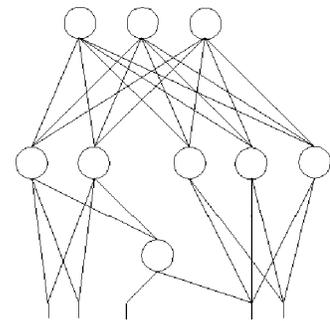


Figure 2. Network Complex

It's easy to build complicated structures by concatenating OWL networks. The complex in Figure 2 gives just a hint of the rich connectivity available — even partial connectivity and recurrence.

This is a feature of all OWL networks, even those with non-local learning laws like backpropagation.

ATOD — A/D converter using an Adaptive BAM as a Hopfield circuit
 BAM — BAM example to enter and receive patterns
 CLN — Competitive learning, with multiple paradigms in one program
 CPN — "Counterpropagation" net illustrates network concatenation
 RABAM — Experiments with the most general ABAM model
 XOR — Backpropagation XOR example

Figure 1. Example Programs