Efficient implementation of neural networks in an object oriented programming environment

Christian Balkenius Robert Pallbo

Cognitive Science
Department of Philosophy
University of Lund
S-223 50 Lund, Sweden

phone: +46+46-108588
E-mail: Christian.Balkenius@fil.lu.se

Keywords: neural networks, object orientation, OODBMS, programming language, implementation techniques, HASP.

INTRODUCTION

In recent years, object oriented programming has gained an increasing interest. Object oriented programming affords easy specification of programs that conform with the model to be implemented. In neural network simulations, objects can be used to make the topology of the network conceptually clear as well as to enhance the comprehensiveness of the program code. However, compilers for object oriented languages do not usually produce as efficient code as compilers for traditional languages. This deficiency is due to the more complex structure of object oriented languages and makes the execution of object oriented programs slower than their traditional counterparts.

It would be hard to design a compiler for an object oriented language that could produce object code that rivals code compiled from traditional languages. In neural network simulations, a small portion of the code is typically responsible for most of the time consumption during execution. If we could optimize these segments of the code, the time factor would not be crucial when selecting a programming language.

In an attempt to overcome the disadvantages of object oriented languages in neural network applications we have designed a language that is object oriented, but allows efficient implementation of neural networks. The language is a part of the HASP environment which includes an object oriented database with persistent objects and various tools for interactive design of graphical user interfaces.

1. GOALS OF DESIGN

The HASP system was designed as an attempt to make specification and simulation of neural networks easier than in traditional languages without loss of efficiency. To make programs easy to understand, we have strived for more rich data structures such as lists, sets and classes. File handling have been simplified by building the system on top of an object oriented data base management system (OODBMS). This makes file handling an integrated part of the HASP programming language. A higher efficiency is accomplished by using high level mathematical operations such as matrix multiplication and set operations.

Portability has been another goal during the design of the environment. Charing of objects and data has been made possible by implementing the system using the OODBMS. Also, the data base is not tied to a specific operating system.

2. THE HASP ENVIRONMENT

The HASP environment consists of five major parts listed below.

• **Object oriented data base management system:** The kernel of the HASP environment. This management system is used for all storage and retrieval of the objects that constitute the HASP system.The OODBMS is used to make all objects in the HASP system persistent, i.e. the objects are automatically retained in between executions.

• **The HASP compiler:** The incremental compiler for HASP code.The compiler makes active use of the database during compilation for storage and retrieval of data structures. Access to the database is included through primitives in the HASP language.

• **The class editor:** An interactive editor for classes and methods which can be edited at any time and linked to other classes and methods in the database. When a class has been changed, the HASP compiler is automatically invoked.

• **The class library:** A set of predefined classes for the management of the database, the visual interface and some common data structures such as trees, lists, sets etc. We have also included a collection of classes for neural network applications.

• **The object editors:** Various editors, each of which is connected to a certain class of objects. The system contains a generic object editor for all classes. This editor can be overridden at each level in the class hierarchy. The default editors use the tools for construction of visual display interfaces, for browsing and editing data structures in the database such as neural network topologies.

3. MODELLING NEURAL NETWORKS

3.1. Neurons

Neurons are modelled by using the class construction in HASP. The use of classes are similar to that in other object oriented languages such as SIMULA and C++. An example of a neuron definition is given in listing 1.

```
class Neuron:
attribute
    out, x: Real;
    weights: array of Real;
method
    Reset: for all w in weights do w := 0;
                  x := 0, out :=0.
main
    Reset.
end.
```

Listing 1. The *Neuron* class definition.

The class definition for *Neuron* contains three parts. The first one specifies the attributes; in this case *out*, *x* and *weights* . The next part describes the methods of the class. Here, *Reset* is the only method and is used to reset the weights of the neuron. The last part of the declaration, the main part, is called once when an instance of the class is created. In this example it is used to call the method *Reset*.

3.2. Networks

Neuron layers are constructed from individual neurons by including them in an array. Arrays in HASP resembles arrays in other languages with the exception of some complex operations. For example, two neuron layers can be joined using array concatenation to form one single layer. It is also possible to extract a subpart of a neuron layer by referencing a partial array.

(a) *Layer* :- *Layer1* : *Layer2*. **(b)** *SubLayer* :- *Layer*[3..5].

Listing 2. Some array operations: (a) concatenation of arrays, (b) assignment of a partial array; a new array is constructed that refers to element 3 to 5 in array *Layer*.

Simulation of a network is achieved by letting the neuron layers operate on each other. The connecting weights are stored in the neurons, but the actual connections are implemented in network modules (Fig 1).

3.3. Modules

When large and complex network are simulated it can be difficult to retain a code that is easy to understand. In object oriented languages the solution is to modularize the model by introducing new classes. The natural approach here, is to define modules for abstract network types. All modules chare some common components, such as inputs and outputs as well as methods to calculate its activity. The class *NeuralModule* is a template for all modules.

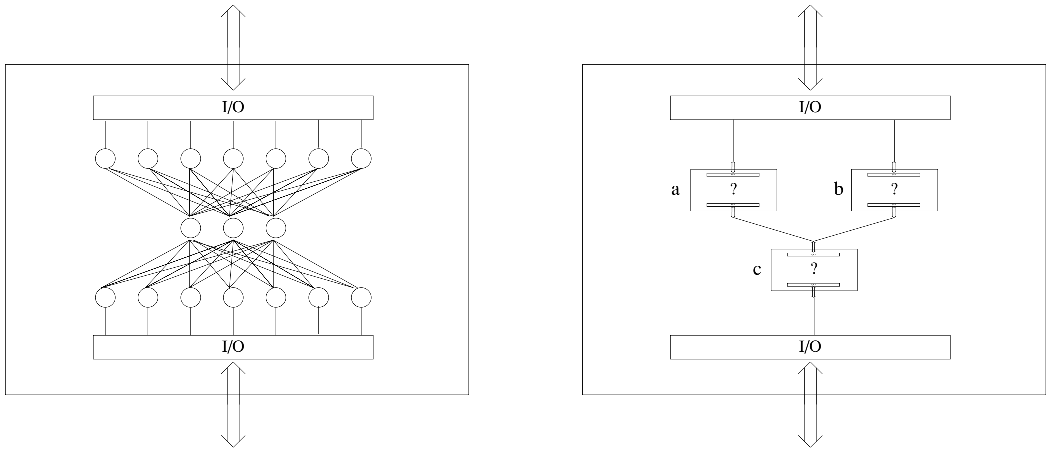

Fig 1. A sample structure of (a) *BackPropModule* and (b) *MultiModule.*

Neural modules can be combined using a *MultiModule* object to form a module in a larger network. A multimodule contains a set of neural modules and a set of connections between them. The modules combined in a multimodule can themselves be multimodules. This allows for a construction of arbitrarily complex network hierarchies. A fundamental part of the HASP environment is the class library. It contains a large set of predefined network modules that easily can be modified or extended.

4. EFFICIENCY

The high level operations in HASP makes the development as well as the execution of network simulations more efficient. Consider for instance the examples in listing 3 using C++ and HASP respectively. This listing illustrates the more compact code of HASP compared to C++.

The HASP example makes use of the subarray construction to access the *out* attributes of all neurons in the hidden layer simultaneously and treat it as a simple array. This array is then multiplied with the weight array in each neuron in the top layer. Note also that HASP make use of dynamic arrays, i. e. the size is determined at run time. An advantage of the approach used in HASP is that the code is described at a higher level relative code written in C++. This makes it possible for the compiler to use more advanced strategies in optimization.

(a) C++

```
neuron  *topNeurons[no_of_topneurons], *hiddenNeurons[no_of_hiddenneurons];
...
for(int i = 0; i<no_of_topneurons; i++){
        topNeurons[i]->x = 0;
        for(int j = 0; j<no_of_hiddenneurons; j++)
        topNeurons[i]->x += topNeurons[i]->weights * hiddenNeurons[j]->out;
};
```

(b) HASP

```
topNeurons, hiddenNeurons: array of neuron.
...
with all topNeurons do
        x := weights * hiddenNeurons[*].out.
```

Listing 3. (a) Calculation of the inputs to a set of neurons using C++. (b) The corresponding calculation in HASP.

Since the integer array *hiddenNeurons*[*].*out* is independent of *topNeurons* over which we iterate, the compiler can optimize the execution by copying the logical real array *hiddenNeurons*[*].*out* into consecutive memory space. In this way it is not necessary to calculate the memory location of the out field in the hidden neurons at every iteration. In the C++ version, this is done m*n times, were m and n are the number of neurons in the top and the hidden layers respectively. In HASP it is done only once for every neuron in the hidden layer, i. e. n times. With a large number of neurons the difference is considerable.

In comparing the execution speed of HASP code and the corresponding C++ code the HASP code was found to be 20% faster. A simple three layer backpropagation net with 40 neurons in each layer was used in the test. The test was run on an IBM AT computer using the Borland Turbo C++ and a prototype of the HASP compiler. We are convinced that more complex networks will make the differences even larger.

4. CONCLUSIONS

The HASP system shows that it is possible to implement neural networks in an object oriented environment without sacrificing efficiency. The high level of abstraction in HASP programming makes the code compact and clear. It is possible to specify neural networks in a modular fashion without any significant effect on execution time. However, construction and modification of complex network structures are much easier. The prototypes that are currently being developed for Unix, PC and Machintosh computers implies that systems of this kind have great advantages.

References

Atkinson M. et al. "The Object-Oriented Database System Manifesto", *Proceedings DOOD 89*, Kyoto, December 1989.

Azema-Barac, M., Hewetson, M., Recce, M., Taylor, J., Treleaven, P., Vellasco, M. (1990) PYGMALION: Neural Network Programming Environment, Proceeding of the INNC-90, Dordrecht: Kluwer Academic Publishers.

Skarra A.H., Zdonik S.B., Reiss S.P. "An Object Server for an Object-Oriented Database", *Proceedings of the International Workshop on Object-Oriented Database Systems*, ACM/IEEE, 1986.

Wand Y. "A Proposal for a Formal Model of Objects", *Object-oriented concepts, databases, and applications* ed W. Kim F.H.Lochovsky, ACM 1989.