NEW TRENDS IN SIMULATION OF NEURAL NETWORKS

Jan Koutnik[1], **Miroslav Šnorek**[1]

[1]Computational Intelligence Group,
Department of Computer Science and Engineering,
Czech Technical University in Prague,
Karlovo nám. 13, 121 35 Praha 2, Czech Republic

koutnij@fel.cvut.cz(Jan Koutnik)

Abstract

In this paper actual simulation techniques and simulation systems for artificial neural networks are compared. We focus on neural network simulators that allow a user easy design of new neural networks. There are several simulation strategies that can be exploited by modern neural network simulators described. We considered the synchronous simulation as the most effective for parallel systems like artificial neural networks. Examples of general simulation systems that can be used for simulation of neural networks are mentioned. Current neural network simulators commonly depend on a type of neural network simulated and cannot be easily extended to simulate a different or a neural network with a brand new architecture and function. Universal simulation tools seem to be suitable for network design but do not support connectionism natively. The missing language constructions and tools for native support of connecting objects in the simulation lead us to design a new simulation tool SiMoNNe - Simulator of Modular Neural Networks, which allows easy design and simulation of neural networks using a high level programming language. The language itself is object oriented with weak type control. It supports native connection of simulated neurons, layers, modules and networks, matrix calculations, easy control of simulation parameters using expressions, re-usability of the result as a source code and more. The language is interactive and allows connection of a GUI to the SiMoNNe core.

Keywords: Artificial Neural Networks, Simulation, SiMoNNe, Programming Languages

Presenting Author's Biography

Jan Koutnik works as a teacher and researcher in Computational Intelligence at Department of Computer Science and Engineering at Faculty of Electrical Engineering at Czech Technical University in Prague. His research is focused on artificial neural networks, self-organization, temporal sequences processing and other methods of computational and artificial intelligence applied in various tasks.

1 Introduction

Artificial neural network simulators are commonly used in experiments with neural networks. There are neural networks of many types – paradigms. We can group neural networks into several categories induced by a history of a neural network research. There were simple neural networks consisting of only one artificial neuron in the beginning [1]. Afterwards, neural networks exploited massive connection of more neurons. Layered neural networks were the main research interest and they are at a high level of importance at present. Modern neural networks are build from hierarchy of modules – neural networks with a high amount of internal connections solving specific tasks. A layered structure is suppressed to the background. The, so called, modular neural networks [2] exploit a decomposition of a task to subproblems and reduced connections by unnecessary full connection between single neurons. The full connection between layers does not automatically lead to the best solution of a particular task. The modularity allows to combine various types of neural networks together. A simulation of such complex system is not usually a simple task.

A realization of neural networks is usually done by their simulation. One can simulate a neural network by a special simulation software, we call it a neurosimulator.

2 Strategies in Neural Network Simulation

Neural networks are parallel systems. It is one of their benefits that makes them suitable for solution of various real domain problems. One must decide how to control parallelism during the simulation. Thus, there are basically four types of simulation strategies in neural network simulation.

2.1 Synchronous simulation

The Synchronous simulation is the most simple strategy. All network units - neurons (layers and modules are expressed by their inner neurons) generate their outputs at synchronous time steps. Each unit calculates its output based on output of other units (and/or itself) from the previous time step. This approach allows simple simulation of recurrent neural networks. An advantage is that we do not need to deal with order of execution of simulated units. The disadvantage is that in some neural network paradigms we expect their output in one time step. This simulation step is called recall of the network - the network is given an input and it recalls the output value. Applying synchronous simulation strategy to e.g. feed-forward neural networks like multi-layered perceptron (MLP) [1] causes that we can either recall each input of the network in one time step and the corresponding output appears on the network output after *n* time steps (while the signal passes through the network layers) or we can consider *n* time steps (equal to number of network layers) as one global simulation step and the simulation of the network is *n* times slower.

The synchronous simulation is suitable for complicated neural networks with high amount of recurrent connections. It is the only one correct simulation strategy when the recurrent connections are not regular and we need to have the network result to be deterministic.

2.2 Data Driven Simulation

Each neuron (considering that layers and modules represent grouping of neurons only) can calculate its activation when proper input is available. This is a typical approach of data-flow architecture. In the field of neural networks this approach is closest to biological systems where each biological neuron fires when a proper input excitement excites the neuron.

A problem appears when a neuron output depends on a recurrent connection. Such recurrent connection causes a *deadlock* because the neuron will wait on its output. This problem can be solved in two ways. First, we will not use this simulation strategy for recurrent neural networks. Second, each recurrent synapse will supply its initial value.

The advantages of the data-flow simulation of neural network are that the current network output reflects it current input. The disadvantage is the recurrent connection problem and its more complicated implementation over the synchronous simulation strategy.

2.3 Topology Analysis Based Simulation

When we know the network topology before the simulation starts, which is a case of most neural networks excluding special neural networks like GMDH networks [3] with dynamic connections, the network topology can be analyzed and exploited.

Neurons in the network are classified into several groups by their distance (measured by amount of connection the signal is passing through) from the network input. In simulation, each group is executed using the synchronous simulation strategy from closest groups to the most distant ones. Recurrent connections are not taken into account during the analysis and they are marked as recurrent.

The advantage is that this strategy properly deals with recurrent connections and in each simulation step the output of the networks corresponds to proper network input. The disadvantage is that a complicated topology analysis has to be performed before simulation starts.

2.4 Other Simulation Strategies

Other simulation strategies for neural networks include:

- **Random permutation**, where each neuron is executed exactly once during the simulation step. The order of the neurons is random. Such simulation strategy is exploited within certain neural networks (e.g. Hopfield network) where it does not affect the network success.
- **Random simulation**, is only hypothetic simulation strategy. The order of activation of neurons

is chosen randomly and it is not guaranteed how much the neuron is executed during the simulation.

- **Predefined order** simulation strategy is given by a user of the network. Such simulation strategy may be usable in simulation of complex recurrent neural network where topology analysis fails and synchronous simulation is not satisfactory.

Concluding the simulation strategies we named several approaches to neural network simulation. We consider that we did not mention strategies for simulation of continuous time neural networks (e.g. spiking neural networks). Such neural networks are usually simulated using synchronous simulation strategy with a short simulation time period. As there is a lot of neural network paradigms, each neural network requires its own simulation strategy. In the field of modular neural networks the synchronous simulation strategy takes place and is the most important one.

3 Neural Simulators

Currently, there is a lot of simulation software and libraries for simulation of neural networks available. A comprehensive introduction to many neurosimulators can be found in [4]. The neural simulators can be divided into several categories.

3.1 Single Purposes Neurosimulators

Single purpose simulators are written especially to simulate one selected neural network paradigm. Such neurosimulator usually contains a graphical user interface, which allows easy manipulation with neural network parameters. The neurosimulator can provide a good understanding to the network behavior. The disadvantage is that it is not easy to simulator another neural network in this neurosimulator. It is almost impossible to do this and the solution is to create another specific neurosimulator for that network.

SOMPAK [5] is an example of such single purpose neurosimulator. It is a set of command-line tools that simulate and visualize function of Kohonen's Self Organizing Map network.

3.2 Multi Purpose Neurosimulators

Multi purpose simulators can simulate more than one specific neural networks. Such neurosimulators have advantages of graphical user interfaces but are more general. The neurosimulators have built in more neural network paradigms. This implies that the core of such neurosimulator is made more general and it may be possible to add neural networks which are not included usually by the means of programming. The disadvantage is that there is no standard of implementation of such multipurpose simulation system and it is also possible that a future neural network is not implementable there because the simulation system has restrictions, which does not affect currently built-in networks.

3.2.1 SNNS

There are several examples of multipurpose neurosimulators. SNNS (Stuttgart Neural Network Simulator) [6] is an open source implementation of a simulator core with GUI. The core is a library written in C. The GUI is either for X-Window or a modern one written in Java usable on all Java capable platforms.

3.2.2 Neural Works Professional

NeuralWare's NeuralWorks Professional is an example of a multipurpose neurosimulator. It contains and can perform simulation of several types of neural networks. Since it is a commercial neural multi purpose neurosimulator a user has to respect and use included neural network paradigms.

3.2.3 Weka and YALE

There are several data mining tools available. Data mining is one of the application field of the neural networks, therefore data mining frameworks include them. A usage of the neural networks performed as a black-box simulation. A user could not study the network interior. The tools are open-sourced. More neural network types can be written and included.

3.3 Universal Simulation Systems

Universal simulation systems which are not directly faced to simulation of neural networks can be also used for the neural network simulation task. Since the neural networks are at most mathematic and statistic tools, one can use systems used for mathematic calculations or tools for simulation of systems. In the field of mathematic calculations we can use Matlab, Mathematica etc.

3.3.1 Matlab

There is the Neural Network Toolbox for Matlab [7] which supplies several neural network paradigms. The toolbox makes a multi purpose neurosimulator from the universal mathematical calculation system. The advantage of Matlab is that it is widely used and if someone is used to use it and if he wants to use a neural network which is already implemented in the toolbox, a usage of the network is easy. More mathematical systems like Mathematica, Statistica, Maple etc. can be used in the same way. The neural network add-on turns it into multipurpose neurosimulator. The generic disadvantage of those systems is a lack of connectionism – the ability to create and connect neurons. The Matlab and Mathematica do not contain explicit language expressions for making connections between simulated units. This is the main reason why we could not consider them as universal neurosimulators.

3.4 VHDL

Due the lack of connection one can use a system which embeds connection subsystem like VHDL [8] for simulation of logical circuits. Description of neural networks using matrix operations is much easier than treating them as logical circuits. There was done a little in the field of neural network simulation using VHDL which may be important from the point of view of their future hardware simulation. In general, description of neural network using hardware terms like signals, buses and architectures leads designer's attention away of the neural network design and usage.

3.4.1 Mathematica

Wolfram's Mathematica is a powerful tool for symbolic mathematics and functional programming. The Mathematica miss a native connectionism and all neural network simulation can be performed using functional programming. A big advantage is an effective syntax and re-usability of the output code that can be used as a source for a next calculation.

3.4.2 Modelica

Another choice can be the Modelica simulation language. It contains built-in connectionism but it is missing neural network library at present. Modelica is an interactive simulator. The compilation is not incremental, compile and run-time are separated. A change of the neural network structure in run-time is not possible.

3.5 Neural Libraries

Neural libraries add functionality needed for simulation of neural networks to common simulation systems and programming languages. We can consider the Neural Network Toolbox for Matlab as such library, but it does not implement connectionism needed for general neurosimulator - it only simulates it. We can take neural network framework named Joone (Java Object Oriented Neural Engine) as an example. It consist of two parts, effective core and graphical user interface (GUI). One can use the GUI or directly exploit the Java core library API. The drawback is that Joone supports layered neural networks only. It is not suitable to build modular neural networks. Joone benefits with parallel processing (or pseudo-parallel by Java threads) of the simulation. This approach eases simulation of layered neural network but such simulation of recurrent modular network with complex structure is not possible due the high recurrence level.

3.6 Universal Neurosimulators

A universal neural simulation system is focused on simulation of simple, layered and modular neural networks. Such organization structures should be enough to simulate various neural networks including modern modular neural networks and their possible improvements. The modular approach is like object oriented approach in classical programming. The module as an object encapsulates other modules and neurons at the lowest granularity level. The layers can be compared to creating arrays in classical programming language. The crucial component that is added by the simulator in comparison with a classical programming language is the connectionism support. As can be seen in following section, it is evident, that the task of creation of an universal simulation system leads to creation of a new programming language, which supports object oriented programming approach (modularity) as well as connectionism.

Search for such universal neurosimulator is not easy. First, we state several features of such simulation system. The ideal neural network simulator has following features:

- **Independence** on its implementation. This means that the simulator does not depend on implementation of any particular framework. This condition is hard to fulfill because there are not any recommendations how the neurosimulator is to be implemented. Neural networks are usually used for data processing, data-mining etc. At present one of the problems of independence is how the data are handled by the neurosimulator and how the neural networks should be stored and interchanged. We have to think about not only the data but about all information that neural simulation requires. In e.g. logical circuits, there are simulation languages like VHDL and Verilog for instance that can be used for data interchange. But in the field of neural networks there is not any standard at present. There are some activities to create data exchange standard for neural networks but those standard does not reflect high evolution of neural networks paradigms. There are XML descriptions for layered networks models but a standard for e.g. modular neural networks is not being developed. That is also because of rapid evolution of neural networks at present. Our neural network language called SiMoNNe, see section 5 can supply solution of the problem especially in the field of modular neural networks.

- **Simplicity**. The neurosimulator should be as simple as possible. It should be easy to implement and simulate current and new types of neurons, synapses, modules and networks. A user has not to be forced to program the networks by himself. A work with the simulator should be effective.

- **Interactivity**. Debugging of a neural network is not an easy task. There is usually a large set of network parameters that require proper setup. The debugging process is a long time work. The simulator should allow easy storage and restore of the simulated network. A user should be able to interactively change the network and reduce the need of restart of the simulation. The simulator should support changing networks (networks which change their behavior themselves), thus it is not possible to use static simulation systems like VHDL and Modelica which do not allow to modify architecture of simulated system in simulation run-time.

- **Modularity and re-usability**. Besides ability of simulation of modular neural networks the neurosimulator should be modular itself. Modular structure of the simulator allows possible embedding into other non-neural applications. All implemented neural network algorithms should be reusable in various parts of the neurosimulator.

3.7 Language versus Graphical User Interface

There are neural network simulators and simulation systems that are driven by a language or a graphical user interface (GUI). Many language driven tools contain a GUI as well. The GUI only simulators can be hardly extended because the GUI structure depends on the simulated neural network in most cases.

The best approach is to control the simulation language by a GUI. The problem is that for high level programming languages a GUI that supports all language features is not easy to construct. In the other words a GUI could not be fully equivalent and as powerful as the underlying programming language.

4 How to Choose a Simulator

Choosing a proper simulator for simulation of modern neural networks is a challenge. One must decide the purpose of the simulation. In case of modern neural networks the final purpose may not be clear at present because nobody knows how much time it takes to perform all experiments before real usage as an embedded neural system.

A choose of the simulator is in general based on an experience of a user. Mathematicians will likely use one of their favorite mathematics tool like Matlab or Mathematica and they will solve a task of converting all those connectionistic models to matrices, functions etc.

People involved in simulation of systems will take systems like Modelica into account but the systems lack of neural network toolboxes and libraries will need more effort to implement all those typical neural network algorithms.

Real programmers will like to use a neural network libraries because they will benefit from experience with well known and high quality debugging tools for common programming languages. The library will supply the connectionism.

Researchers focused on a specific task as well as specific neural networks will look for specific simulators for a particular networks but they may have problems with further embedding of the network. The will benefit from a particular graphical user interface and well working simulator but problems with extending of their neural network paradigm will appear in a serious research.

There is a simulator's matrix in Table 1 which gives an overview of common simulation systems from the point of view of neural networks simulation.

5 SiMoNNe – Simulator of Modular Neural Networks

Motivated by a design of new neural network architectures we were forced to design and implement a new neural network simulation system, which is enough powerful to simulate a neural network of an arbitrary structure and function and serves a user neural network specific features, especially the connectionism. We named it Simulator of Modular Neural Networks (SiMoNNe) [9].

The SiMoNNe consists of high level programming language as well as the language interpreter and the simulation workspace. Considering the criteria for ideal neural simulator the SiMoNNe has following features implemented.

It uses modules (objects) to build modular neural networks. The modules can be connected as a user decides using simple expressions that can aggregate full connections between modules and layers. The simulator is interactive, basic interaction is done using command line and interactive compiler of the SiMoNNe Language. The simulation system is universal and paradigm independent. It can simulate arbitrary neural network. The language allows to perform matrix operations and other mathematic calculations.

The SiMoNNe supports synchronous simulation strategy and predefined order strategy natively. Those are the most important strategies for modular neural network simulation. Other simulation strategies like random order and permutation can be easily implemented within SiMoNNe Language. Topology analysis is not supported due the dynamic and interactive character of the simulation. There is no starting (before compile) point when the topology analysis can be performed.

The SiMoNNe shortens description of common neural networks to minimum size of code. The abilities of the language were tested on common neural networks like Multi-layer Perceptron, Hopfield and Kohonen type networks as well as on new modular neural networks like CALM (Categorizing and Learning Modules), TICALM (Temporal Information Categorizing and Learning Map)[10] and their variants.

The main features of the SiMoNNe are:

- **connectionism** is implemented by basic language expressions. A user can easily connect neurons, layers and any arbitrary defined structures and subnetworks,
- **modularity** is expressed by object oriented programming. Modules and network are created as objects,
- **language** driven simulation. SiMoNNe is a programming language and it's incremental compiler and interpreter,
- **GUI** accompanies the SiMoNNe. GUI is connected to the simulation core using the SiMoNNe Language,
- **bi-directionality** of the SiMoNNe Language assures that a simulation output is a code in SiMoNNe Language syntax as well,
- **matrix calculations** are embedded in the language,

Tab. 1 Simulator Matrix

Feature / System	Matlab	Mathematica	Modelica	Joone	VHDL	SiMoNNe
Neural Extensions	Yes	Yes	-	Yes	-	Yes
Native Connectionism	-	-	Yes	-	Yes	Yes
Matrix Calculations	Yes	Yes	-	-	-	Yes
Modular Networks	-	-	Yes	-	Yes	Yes
Modularity	-	-	Yes	Yes	-	Yes
Embedding	Limited	-	Yes	Yes	-	-
Implementation Independence	-	-	-	-	-	-
Simplicity	-	-	-	Yes	-	Yes
Universality	Yes	Yes	Yes	-	Limited	Yes
Interactivity	Yes	Yes	-	GUI	-	Yes

- **query subsystem** gives a user a tool for querying all objects, neurons, modules and their internal values by expressions,
- **collective operations** with arrays and matrices allow easy and rapid setup and data acquisition from a simulation,
- **universality** assures that almost any network that can be simulated using synchronous simulation can be simulated in SiMoNNe. The SiMoNNe is a neural network paradigm independent simulator. Neural networks and their simulations are described by programs in the SiMoNNe Language.

6 Conclusion

Based on an unsuccessful survey and search for an ideal neural network simulator we stated several requirements that we put on such simulation system.

We found that future of the neural network research could not be satisfied by existing single purpose simulators when dealing with a problem which neural network is suitable for solution of particular task. Such interactive neural simulators are not exploitable as embedded systems as well.

First, we need a simulation system which is close to the ideal neural network simulator for experiments with modern modular neural networks, thus we designed SiMoNNe - Simulator of Modular Neural Networks. SiMoNNe allows easy definition, creation, execution and debugging of modular neural networks of various structure, topology and function. It supports network with heterogeneous neurons and modules. One can play with connections within modules, replace units and change their behavior on the fly and see a response of the simulator at the moment.

Acknowledgment

This work is supported by the research programme “Transdisciplinary Research in the Area of Biomedical Engineering II” (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic. SiMoNNe can be downloaded from `http://cig.felk.cvut.cz/projects/simonne`.

7 References

[1] Haykin S., *Neural Networks, A Comprehensive Foundation*, IEEE Press, 1994, ISBN 0-02-352761-7

[2] Ronco E., Gawthrop P., Modular Neural Networks: State of the Art, Technical Report CSC-95026, Center for System and Control, University at Glasgow, UK, 1995.

[3] Kordík P., Náplava P., Šnorek M., Genyk-Berezovskij M., The Modified GMDH Method Applied To Model Complex Systems: *ICIM 2002 Conference Proceedings*, Lviv, Ukraine, 2002

[4] Murre J. M. J.: Neurosimulators. In: Arbib M.A., The Handbook of Brain Theory and Neural Networks, The MIT Press, 1998, pp. 634-639.

[5] Kohonen T., Hynninen J., Kangas J., Laaksonen J.: SOM PAK: The Self-Organizing Map program package, Report A31, Helsinki University of Technology, Jan. 1996.

[6] Zell A., Mamier G. Vogt M., Mache N., Hübner R., Döring S.: Stuttgart Neural Network Simulator, User Manual, Version 4.1, 1995.

[7] Demuth H., Beale M.: Neural Network Toolbox, User's Guide, Version 4, 2001, Available at `http://www.mathworks.com`

[8] ANSI/IEEE Std 1076-1987, *IEEE Standard VHDL Language Reference Manual*, New York, USA, 1998

[9] Koutník J., Šnorek M., Efficient Simulation of Modular Neural Networks, *In Proceedings of the 5th EUROSIM Congress Modelling and Simulation*, EUROSIM-FRANCOSIM-ARGESIM, 2004, vol. II, ISBN 3-901608-28-1.

[10] Koutník J., Šnorek M.,Single Categorizing and Learning Module for Temporal Sequences, *IJCNN 2004 Conference Proceedings*, Piscataway: IEEE, 2004, p. 2977-2982. ISBN 0-7803-8360-5.