DOI: 10.3745/KIPSTD.2009.16-D.1.139

인공 신경망의 시냅스 가중치 관리용 도구 개발

신 현 경†

요 약

다양한 기계 학습 이론을 총체적으로 구현할 수 있는 포괄적 체제로서의 신경망은 현재 활용되는 기능보다 더 큰 잠재력을 지니고 있다. 신경망의 여러 가지 특성 가운데, 연상 기억 능력을 자연적으로 활용 할 수 있는 신경망 내 시냅스 고유의 구조적 속성이 신경망의 가장 중요한 특성이다. 그러나 이론적 장점에도 불구하고, 네크워크의 복잡성에 기인한 다양한 형태의 피할 수 없는 난제들로 신경망의 실제적 구현 및 유지의 어려움이 잘 알려져있다. 본 논문에서는 인공 신경망의 시냅스 가중치 관리를 효과적으로 관리 할 수 있는 도구를 설계 및 구현 하였다. 개발된 소프트웨어는 다양한 형태의 신경망들의 훈련 단계에서 신경망 내 시냅스의 가중치 변화를 표시해 주는 기능을 갖추고 있다.

키워드 : 컴포넌트 기반 소프트웨어 공학, 그래픽 사용자 인터페이스, .NET, 인공 신경망

Development of Monitoring Tool for Synaptic Weights on Artificial Neural Network

Shin Hyun Kyung†

ABSTRACT

Neural network is a very exciting and generic framework to develop almost all ranges of machine learning technologies and its potential is far beyond its current capabilities. Among other characteristics, neural network acts as associative memory obtained from the values structurally stored in synaptic inherent structure. Due to innate complexity of neural networks system, in its practical implementation and maintenance, multifaceted problems are known to be unavoidable. In this paper, we present design and implementation details of GUI software which can be valuable tool to maintain and develop neural networks. It has capability of displaying every state of synaptic weights with network nodal relation in each learning step.

Keywords : CBSE, GUI, .NET, Artificial Neural Network

1. Introduction

Since *Turing Machine* has been proposed, computability has been one of the central subjects in computer science [1]. As a part of the key approaches to computability issues, machine learning technique can be characterized by its effort on self learning capability [2]. Among various machine learning techniques, the neural network is a framework on which various computational machine learning methods are based. The neural networks utilize the concept of network composed of the computing nodes and the links among them [3, 4, 5]. As the most significant characteristic of neural networks, they do not follow the fetch-execute-store process of the von Neumann digital computer; on the other hand, they globally respond to input pattern stimuli [6]. In other words, they do not contain data and algorithmic instructions in a separate memory system; instead they store data throughout the network in the pattern of weights, interconnections and states of the neurons. This reflects highly problem-dependency property of neural network programming. This property also often discourages neural network development from the fact that structural complexity of network makes the verification of states very hard.

Our focus in this paper is on design and implementation of the neural network in aspect of software development. From the software engineering point of view, computation with the neural network can be considered as construction of pipelines connecting four individual modules: feature vector selection, specification of learning algorithms, control mechanism, and user interface. With support of the modern programming

※ 이 연구는 2008년도 경원대학교 지원에 의한 결과임.
† 종신회원 : 경원대학교 수학정보학과 조교수
논문접수 : 2008년 11월 19일
심사완료 : 2008년 12월 5일

languages based on object-oriented design, the elements of neural network structure, node and synapse, are easily formulated to the low-level computational units. We also adopt the usage of hierarchical layered structure composed of the neurons and the links in order to utilize component based software engineering.

The contents of the paper are organized as follow: in section 2, previous research works related are investigated, in section 3, the proposed design of GUI developed in this paper is explained, in section 4, implementation details of GUI are described, in section 5, experimental results can be found and the conclusions are also discussed.

2. Related Works

Among the four topics on design and implementation of the neural networks mentioned above, the feature vector selection is exclusively problem dependent, the learning algorithms devised to train neural network are fairly standardized by the products of *MatLabTM* [7] and Intel's *OpenCV*[8, 9], and the control mechanism and the user interface are not standardized and less problem dependent. Therefore, the main discussion in this paper is devoted to development of graphic user interface.

Neural network computation requires fairly small counts of control parameters. However, it requires determining the value for each synaptic weight, which usually takes tens of hundreds, through training method. Problem of training lies in the fact that update of synaptic weights is data dependent. Misra [10] surveyed on implementation of ANN as a part of project to understand inordinate time to train or run ANN; Kim and Lee [11] explored synaptic parameter space to study temporal aspect of neural information processing.

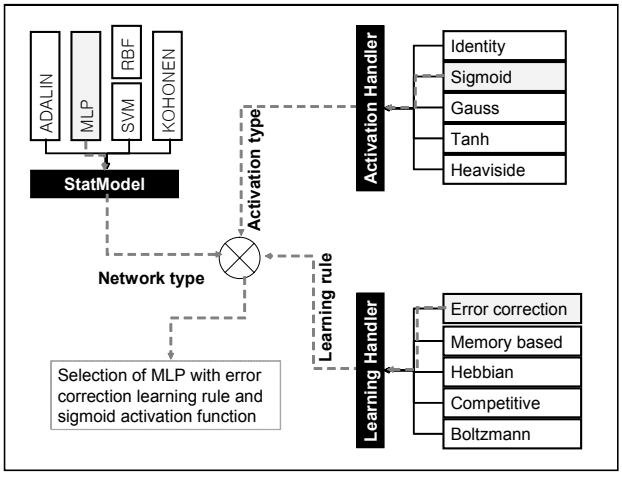
Many researchers tend to use the existing commercial or the open source libraries for design and simulation of neural networks, e.g., Neural Network ToolboxTM of MatLabTM and Intel OpenCV. Advantage of employing existing library is mainly twofold: fast development cycle and reliability of performance. Both the machine learning library of OpenCV and the neural network toolboxTM also provide the facilities for graphic user interface. In order for successful simulation, strong data dependency of neural network programming inevitably requires capability of thorough investigation on synaptic values at each training stage. However, the existing GUI has limitation of flexibility. We developed a programming library of various types of neural networks from scratch and also created GUI as a means to improve the library.

3. Design

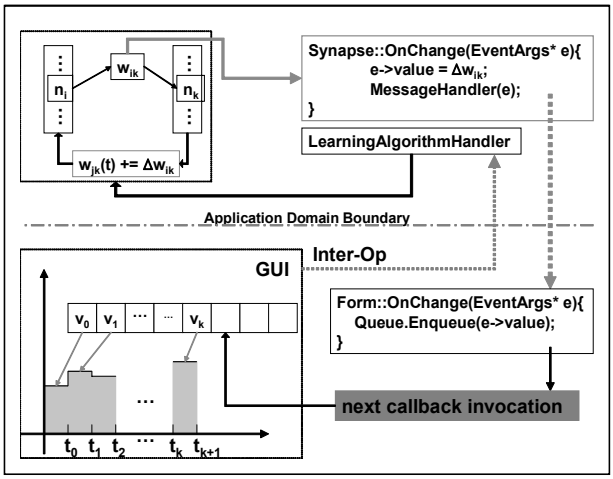
As a part of project for development of computer vision library, the machine learning technologies have been developed on framework of neural networks including the three basic functionalities of statistical modeling such as classification, regression, and clustering. In software engineering perspective, we followed a recent trend of design pattern compatible with component-based software engineering(CBSE)[12, 13]. (Fig. 1) illustrates the underlying idea of component-base engineering applied to construct our neural network library.

In component-based engineering of neural network library, we consider the following three subjects as infrastructural bases: network type; adopted learning algorithm method; and activation-function type. Diverse types of neural networks - ADALIN, MLP, KNN, SVM, SOM, and etc - are under the hierarchy of object modeling where a base class 'StatModel' serves a top parent object to all network objects. Various types of learning algorithms are designed to have a common interface so that they are unified by a generic delegate (function pointer) object and they are parameterized by user-specific input value. Unlike conventional neural networks, in our neural network library, network model and learning algorithm are designed as mutually orthogonal. Several activation functions are also designed to have a common interface. Moreover they are devised to be independent on learning algorithms. In (Fig. 1) overview of neural network library shows component-based design pattern. The gray dashed lines illustrate component-wise construction of the neural network of interest.

The base elements of neural network are neuron and synapse. From the design point of view, neuron is consisted of three fields: a memory block to hold value, two memory



(Fig. 1) Design of the underlying neural network library



(Fig. 2) Design of the callback routines to pass the changes occurred in synapse states

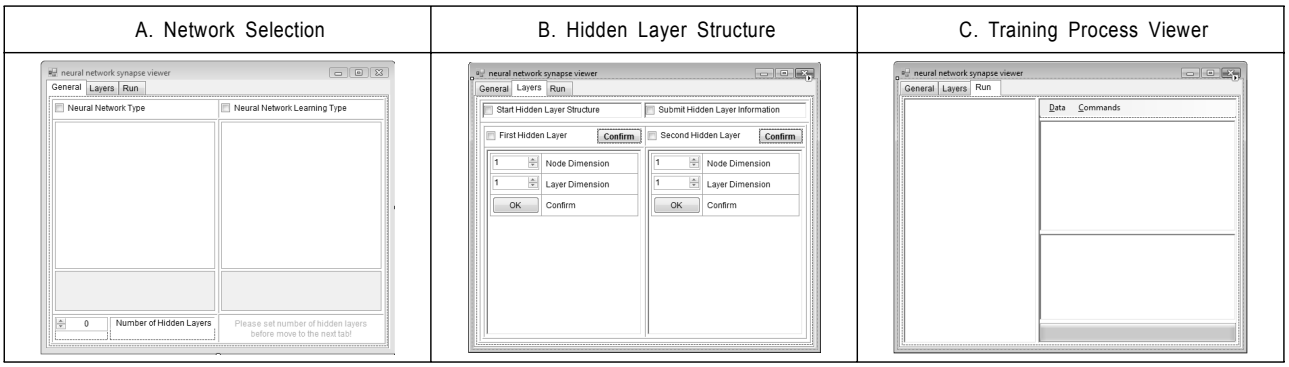
blocks to represent addresses of pre- and post- synapses. Synapse is consisted of three fields: a memory block to hold value(synaptic weight), two memory blocks to represent addresses of pre- and post- neurons. The most significant aspect of element-based design is that we can easily introduce a layer of neurons and a layer of synapses so that the neuron and the synapse objects can keep its simplest form while the layer class takes charge of very complex functionalities including forwarding, application of activation, back-propagation if necessary, etc. Structural simplicity of neuron and synapse objects helps to analyze progressive changes occurring during performance of neural network. We adopt callback technique [14], shown in (Fig. 2), prevalent in CBSE and event driven architecture in .NET language.

4. Implementation

As seen in (Fig. 3), the GUI is consisted of the three divided tabs: network type selection tab; network layer structure selection tab; and presentation tab. The first tab is consisted of three separated panels: at the left panel a user can select the type of neural networks to perform simulation, at the right panel a user can select the type of learning rules to be adopted by the neural network, and at the bottom left panel the number of hidden layers can be specified. At the second tab structural information on the hidden layer, if it exists, is received. Due to the fact that a number of hidden layers must be decided at run-time, the panels used in the second tab are designed as a form of the user control and not observable. Only the panels selected are dynamically allocated and exposed to user to set structural information of each hidden layer. The third tab is consisted of the three panels, the progress bar, and the one menu item. Users can link the training data to the network through the menu item. Three panels are located at left, top-right, and bottom-right. The left panel is designed to show the progressive values of the synapses defined in the neural network. For compatibility with run-time decision on the number of synapses defined in the network, each synaptic value viewer is designed as user-control and is located automatically with identifiable index. The top-right panel is designed to present structure of the neural network. The bottom-right panel is designed to show messages invoked from the program library.

4.1 User Inputs for Network

In our system neural network is considered as an object which can be parameterized by the two major options ('Neural Network Type' and 'Learning Type'), the several minor options (such as 'Activation Method', 'Connectionism', 'Error Estimation Type', and 'Training Parameters'), and the number of hidden layers. To achieve this model in software level, our GUI has been implemented to have capability of creating several different types of neural networks at run-time. First of all, neural network type can be selected among the following options: ADALINE, MLP, RBF, SOM, SVM, Committee-Machine, Recurrent-



(Fig. 3) Snap shot of GUI consisted of three layers

Networks, and etc. It is conventional that a learning rule depends on a neural network type such as MLP with error correction, and SOM with competitive learning, etc. However, in our GUI, network type and learning rule can be constructed as marginally orthogonal. Even though it is not always recommended, through our GUI, it is possible to create a neural network with any type of learning rule, which is useful for design of new style of learning machine. Secondly, selection of learning rule is parameterized with possible options of 'error correction', 'memory based', 'competitive', 'regularization', 'hebbian', and 'boltzmann'. The 'error correction' is designed for back-propagation style machine learning[5]. The 'memory based' is used in associative memory style machine learning such as k-NN[15]. The 'competitive' is used in un-supervised machine learning such as Kohonen SOM [16]. The 'regularization' is used in regression style machine learning such as RBF [17]. The 'hebbian' is typically used in Hopfield network and has two sub-choices 'Hebbian-Apex' and 'Hebbian-Gha' [18]. The 'Boltzmann' is used in recurrent style machine learning such as Recurrent-Dynamics and generative Hopfield network[19, 20].

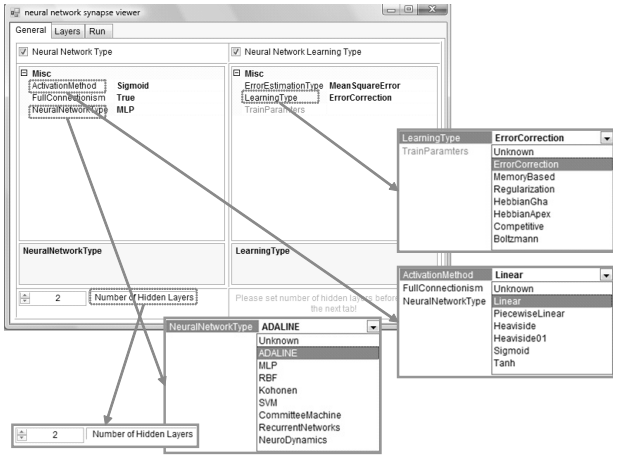
Selection of the user inputs is illustrated in (Fig. 4) For the convenience of visualization, only four important options are displayed.

Selection of the number and the dimension of neurons to be assigned in each layer is straightforward. For simplicity of paper we omit to show details on the second tab which is about user input for hidden layer structure.

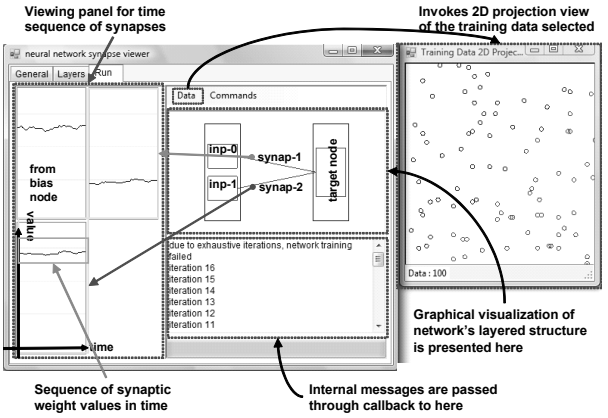
4.2 Presentation Layer

Our purpose of this research is to develop software visualizing the interior states of neural networks in temporally progressive manner. This is achieved in the presentation layer where the dynamic structure of neuron layer and the temporal states of synaptic weights are visualized as well as the messages invoked from the network. The progress bar is also presented to indicate the current stage of network training. The callback technique, described in (Fig. 2), is applied to send state values from synapses and to create routines for message passing. In (Fig. 5), presentation layer is shown when ADALINE is trained with sample data. Distribution of the sample data is appeared at the right side of main presentation panel. At the left section of main panel, values on the three synapses(one from bias node, two from input node).

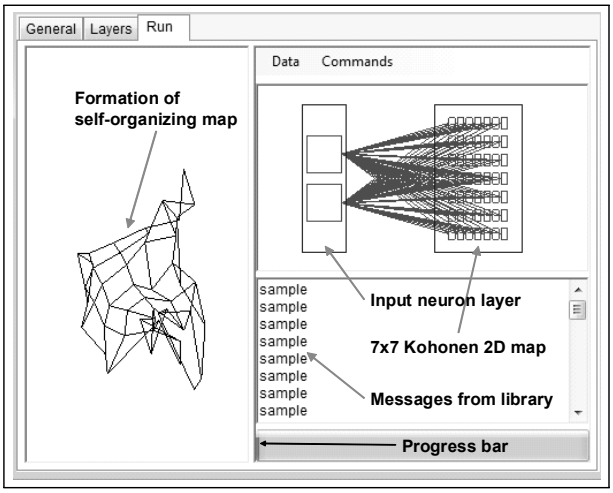
In (Fig. 6), training stages are displayed when Kohonen network is selected to train. The synapses connected to 7x7 2-D Kohonen map(layer) start to show self-organizing property. In this figure SOM is very early stage. As seen in (Fig. 5), top-right panel displays network structure (which is mostly stationary) and bottom-right panel contains



(Fig. 4) User inputs on the first tab for neural network creation



(Fig. 5) Snap shot of monitoring the training process with ADALINE



(Fig. 6) Snap shot of monitoring the training process with Kohonen network

messages invoked and passed from the internal library. However, in left panel ordering of synapse value at a given time instead of time signal graph.

5. Discussion and Comparison with Other Approaches

Intel *OpenCV* and *MatLab* Neural Network ToolboxTM are regarded as the standard products in machine learning research. Regardless of library performance, we compare the GUI related functionalities, which can be seen at <Table 1> As API libraries, all three provide the functions for creation, training, and simulation using neural networks. However, Intel *OpenCV* does not support GUI while *MatLab* toolbox and our library support it. In perspective of flexibility, Intel and our suggested library provide user callback interface while *MatLab* doesn't. The neuron models adopted are almost similar but our library has more flexibility of generalization as allows using arbitrary dimensional neuron node. Available transfer functions are also similar. In terms of availability of neural network models, *MatLab* provides the most. For training style, *MatLab* is superior as it supports sequential input. Creating networks from GUI is supported by *MatLab* and our library. Recent release (6.0.1., at Oct. 2008) of *MatLab* toolbox supports the functionality of monitoring network training process which is also a main feature of our library. However, we display more detailed information than *MatLab*. Our library displays the values stored in each synapse and neuron while *MatLab* displays the status of training. In aspect of programming portability, modularity and callback usability are important issue. For modularity, Intel library does not have solid interface design while *MatLab* and our library are designed to consider modularity. *Matlab* provides the toolboxes while we provide a component library. Consequently, our library gives more flexibility to the users since the component library allows user specified callbacks.

With the aid of the GUI developed, controllability over internal parameter settings is greatly enhanced for neural network's learning process. For example, we could observe (in run-time) repetitive fluctuation pattern in sequence of synapse values when we performed training ADALINE with sample data having very obscure decision boundary as seen at the right side panel in (Fig. 5). Then, when we moved on to use MLP, we could easily observe effects of the numbers of neurons assigned in hidden layers and the selection of activation function type. For other cases, it was critically useful when we were investigating Kohonen's competitive learning process. We could come up with the appropriate iteration numbers to determine ordering and converging phases. The number of clusters as pre-process to RFB learning is one of the major concerns in neural network studies. We could conveniently control the parameters to produce an appropriate number of clusters by analyzing the pre-process by observing the changes in the synapses.

Our future plan is to generalize our GUI's interface to be compatible with other commercial or open source neural

〈Table 1〉 Comparison among software

			Intel *OpenCV*	Neural Network Toolbox™	Our library
Supporting functionalities	API		creation, training, and simulation	creation, training, simulation, GUI	creation, training, simulation, GUI
	user callbacks		support	No support	support
Modeling	Neuron Model	Node	1 dim	1 dim	arbitrary
		Transfer functions	Linear, sigmoid, hard limiter, tanh	Linear, hard limiter, sigmoid	Linear, sigmoid, hard limiter
	Network Model		MLP, SVM	ADALIN, MLP, RBF, SOM, Layer-Recurrent	ADALIN, MLP, RBF, SVM, SOM
	Training Style	Input style	Concurrent Input	Concurrent input, sequential input	Concurrent input
		mode	Batch training	Incremental and batch training	Incremental and batch training
GUI	Create networks		N/A	Perceptron from GUI	Create network from GUI
	Training feedback		N/A	Display training status	Display the value of each synaptic weight
Portability	Modularity		Fair	Good	Good
	Program language		C/C++	MatLab	C++/.NET

network libraries such as MatLabTM and Intel OpenCV [20].

References

[1] Turing, A. M., "Computing machinery and intelligence," Mind 50: 433-460, 1950.

[2] Sipser, M., "Introduction to the Theory of Computation," 2nd edition, Course Technology, Boston, 2005.

[3] Bishop, C. M., "Pattern Recognition and Machine Learning," Oxford University Press, Oxford, UK, 2007.

[4] Hassoun, M. H., "Fundamentals of Artificial Neural Networks," The MIT Press, Cambridge, MA, 1995.

[5] Haykin, S., "Neural Network and Machine Learning," 3rd edition, Prentice Hall, Upper Saddle River, NJ, 2008.

[6] Caudill, M., and Butler, C., "Naturally Intelligent Systems," The MIT Press, Cambridge, MA, 1992.

[7] http://www.mathworks.com/products/neuralnet/

[8] http://sourceforge.net/projects/opencvlibrary/

[9] Bradski, G.; Kaehler, A., "Learning OpenCV: Computer Vision with the OpenCV Library," O'Reilly, Cambridge, MA, 2008

[10] Misra, M., "Parallel Environments for Implementing Neural Networks," Neural Computing Surveys, 1:48-60, 1997.

[11] Kim, S., and Lee, S., "Phase Dynamics in the Biological Neural Networks," Physica A, 288:380-396, 2000.

[12] Szyperski, C., "Component Software: Beyond Object-Oriented Programming. 2nd ed.," Addison-Wesley Professional, Boston, MA, 2002.

[13] Heineman, G. T., and Councill, W. T., "Component-Based Software Engineering: Putting the Pieces Together," Addison-Wesley Professional, Reading, MA, 2001.

[14] Stroustrup, B., "The C++ Programming Language, third Edition," Addison-Wesley, Reading, MA, 1998.

[15] Shakhnarovish, Darrell, and Indyk, "Nearest-Neighbor Methods in Learning and Vision," The MIT Press, Cambridge, MA, 2005.

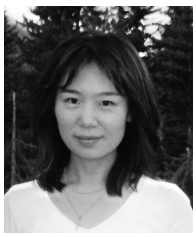
[16] Kohonen, T., "Self-Organizing Maps", 2nd edition, Springer-Verlag, Berlin, 1997.

[17] Buhmann, M. D., and Ablowitz, M. J., "Radial Basis Functions: Theory and Implementations," Cambridge University Press, London, 2003.

[18] Paulsen, O., and Sejnowski, T. J., "Natural patterns of activity and long-term synaptic plasticity." Current opinion in neurobiology 10 (2): 172 - 179. 2000.

[19] Hinton, G. E., Osindero, S., and Teh, Y., "A fast learning algorithm for deep belief nets," Neural Computation, 18: 1527-1554.[1], 2006.

[20] Raphael, R., "DARPA Urban Challenge, a C++ based platform for testing Path Planning Algorithms: An application of Game Theory and Neural Networks," Cornell University, 2007.



신 현 경

e-mail : hyunkyung@kyungwon.ac.kr

2002년 State University of New York (Stony Brook) 대학원 응용수학과 (공학박사)

2008년~현 재 경원대학교 수학정보학과 조교수

관심분야 : Neural network. Machine learning. Image processing