Creating of Conceptual Lattices using Multilayer Perceptron

Vladimír Havel, Jan Martinovič
Václav Snášel and Karel Vlček

Department of Computer Science, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
vladimir.havel@vsb.cz, jan.martinovic@vsb.cz, vaclav.snasel@vsb.cz,
karel.vlcek@vsb.cz

Abstract. Creating of concept lattices is unfortunately difficult time-consuming process. Neural networks can solve this problem, because they are adaptive in nature, due to their use of a training phase for learning the relationships in the input data, and their generalization capabilities provide a means of coping with noisy or incomplete data.
In this text there is presented one possible approach to creating conceptual lattices using multilayer perceptron. The applicability of this approach was experimentally tested as well as parameters it depends on. Hardware acceleration (of the computing) is also discussed.

1 Introduction

Concept analysis has proven to be a valuable tool for gaining insight into complex data. In many applications of concept analysis experts learn from formal contexts by inspecting their carefully laid out concept lattices. Contexts in these applications tend to have a modest size.

The algorithmic complexity of concept analysis for these applications is consequently a minor concern. But concept analysis is used increasingly for applications like program analysis inside a compiler or is combined with statistical analysis where large contexts are constructed by a program. The resulting concept lattice is no longer inspected visually but is part of an application's internal data structure. For these applications the algorithmic complexity of concept analysis does matter. This paper presents a solution for computing the concept lattice by neural network. The neural network will be implement using hardware accelerator.

2 Background

2.1 Formal Concept Analysis

FCA has been defined by R. Wille and it can be used for hierarchical order of objects based on object's features. The basic terms are formal context and formal concept. In this section there are all important definitions the one needs to know to understand the problems.

Radim Bělohlávek, Václav Snášel (Eds.): CLA 2005, pp. 149–157, ISBN 80–248–0863–3.

Definition: A formal context $C = (G, M, I)$ consists of two sets G and M and a relation I between G and M. Elements of G are called objects and elements of M are called attributes of the context. In order to express that an object g is in a relation I with an attribute m, we write gIm and read it as "the object g has the attribute m". The relation I is also called the incidence relation of the context.

Definition: For a set $A \subset G$ of objects we define

$$A^{\uparrow} = \{m \in M \mid gIm \; for \; all \; g \in A\}, \tag{1}$$

the set of attributes common to the objects in A. Correspondingly, for a set $B \subset M$ of attributes we define

$$B^{\downarrow} = \{g \in G \mid gIm \; for \; all \; m \in B\}, \tag{2}$$

the set of objects which have all attributes in B.

For more information about formal concept analysis see [5]

2.2 Multilayer Perceptron Overview

Multilayer perceptrons are the most used neural networks, because they are universal approximators.

General properties of artificial neural networks are:

- Many simple neuron-like threshold switching units
- Many weighted interconnections among units
- Learning to associate inputs to output by adjusting the connection weights (adaptation phase)
- Ability of generalization
- Highly parallel, distributed processing
- Fault tolerance

The function of formal neuron in the network can be described as follows:

$$y = f(\xi) = f\left(\sum_{i=0}^{N} x_i w_i\right). \tag{3}$$

where x_i is the i^{th} input, w_i is a synaptic weight of the i^{th} input, N is the number of inputs, and f is (usually nonlinear) *transfer function*.

As transfer function f was used sigmoidal (logistic) function because it is widely used in the many applications:

$$f(\xi) = \frac{1}{1 + e^{-\lambda \xi}}, \tag{4}$$

where λ represents the *slope* of the sigmoid.

For adaptation of MLP was used classical *backpropagation algorithm* (BP) [7] [16]. The backpropagation algorithm is a gradient descent method which works in much the same way as the name suggests:

After propagating an input through the network, the partial error (5) is calculated and the error is propagated back through the network while the weights are adjusted (according to (6)) in order to make the error smaller.

$$E_k(\mathbf{w}) = \frac{1}{2}\sum_{j\in Y}(y_j(\mathbf{w},\, \mathbf{x}_k) - d_{kj})^2. \tag{5}$$

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}}, \tag{6}$$

where y_j is the calculated output and d_{kj} is the desired output of neuron j, k is the number of learning pattern, w_{ij} is the synaptic weight between the i^{th} neuron of lower layer and the j^{th} neuron of higher layer, η is a *learning rate*.

3 Applying MLP to conceptual lattices

The reason why use neural network is obvious: fault tolerance, generalization, adaptability, etc. Fault tolerance is one of the key attributes of neural networks, and this accrues from the distributed representation nature of neural networks. Because the information is stored in a distributed way throughout the electronic (or other) implementation, even if a few percent of the internal devices fail, the impact on the overall operation can be negligible. In local representation devices, failure of internal components means total absence of the pieces of information that were stored in. Also, for the specific test here considered relative to the conceptual lattice, it appears, there will be significant speed and memory size advantages on the side of neural networks vs. parallel hardware.

There is a number of works dealing with Neural network approaches to conceptual lattices, graphs etc. So far Self-organizing map or a type of recurrent neural network (like BAM and Hopfield) was used [10, 2, 9].

Application of neural networks to any problem context requires that the data associated with that problem be encoded (represented) in a manner meeting specific requirements imposed by the neural-network methodology. We used the (reduced) matrix representation of conceptual lattices.

3.1 Representation of concepts for neural network

The network has 3 layers:

1. input layer – attributes
2. hidden layer
3. output layer – objects

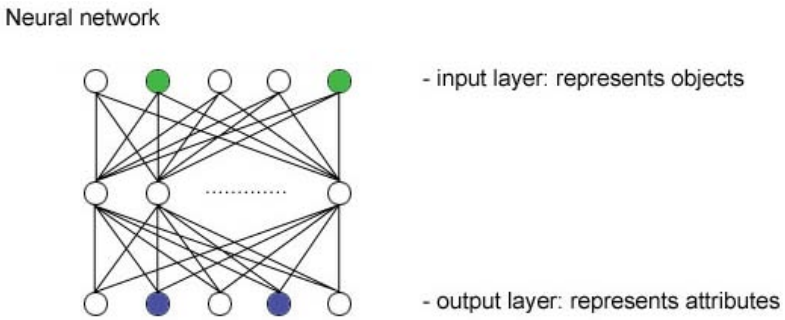

Fig. 1. Example of MLP recognizing concepts.

So the MLP realizes a mapping (i.e. a function) τ:

$$\tau : \{A\}^m \rightarrow \{O\}^n, \tag{7}$$

where A is the set of attributes, and O is the set of objects. $m, n \in \mathbb{N}$.

The principle of applying MLP to conceptual lattices is as follows:

MLP will learn selected patterns (composed of attributes together with the required response objects).

The purpose is let the MLP recognize hidden dependencies among attributes and objects during the adaptation phase. After learning can the MLP compute objects (de facto concepts) even for non-learned configuration of attributes.

MLP in active phase gives response in real-time.

For real applicability we need solve two non-trivial problems:

1. How to select representative training set. We need select the patterns which contains the important features of concepts. The quality of generalization of MLP is very sensitive to this selection.
2. The speed of adaptation.

3.2 Preliminary Experimental results

The testing set contains all concepts which were making from generated collection of 99 documents. Sample of documents is below:

document 20: Xbbbba Xbbbbb Xbbbbc Xbbbbd Xbbbbe Xbbbbf Xbbbbg bbbbh bbbbi bbbbj Xffffa Xgggga Xggggb Xggggc Xhhhha Xhhhhb Xhhhhc Xhhhhd Xhhhhe

document 21: Xbbbba Xbbbbb Xbbbbc Xbbbbd Xbbbbe Xbbbbf Xbbbbg Xbbbbh bbbbi bbbbj Xgggga Xhhhha Xhhhhb Xhhhhc Xiiiia Xiiiib Xiiiic Xiiiid Xiiiie

document 22: Xbbbba Xbbbbb Xbbbbc Xbbbbd Xbbbbe Xbbbbf Xbbbbg Xbbbbh Xbbbbi bbbbj Xhhhha Xiiiia Xiiiib Xiiiic Xjjjja Xjjjjb Xjjjjc Xjjjjd Xjjjje

The testing set consists of 890 concepts in context with 150 attributes and 99 objects. The training set for neural network was composed by each fifth concept from context (178 inputs).

We performed a number of experiments on network and BP parameters like topology, learning rate, number of iterations and so on.

Initially it was designed the network topology $190 - 99 - 99$ in order to reach good generalization results. The best results, (i.e. the minimum number of wrong identified concepts (objects) in the whole set of concepts) gave network with $190 - 150 - 99$ topology in conjunction with small learning rate – 0.01, which needed over 2000 iterations (on an average).

4 Hardware acceleration

Since it is possible to have for example matrix of type 300000×300000 in the real world, and time of adaptation is increasing rapidly, the fast computing device is needed.

Three ways of efficient implementing of digital neural computing is described in this section.

4.1 Reducing data flow

A better way of handling large matrices is illustrated by the following small example [8]. Let us assume two 3×3 matrices to be multiplied, and a device with six inputs for different values. It is possible to reverse order of performing multiplication:

$$\mathbf{AB} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} \sum\limits_i a_{1i}b_{i1} & \sum\limits_i a_{1i}b_{i2} & \sum\limits_i a_{1i}b_{i3} \\ \sum\limits_i a_{2i}b_{i1} & \sum\limits_i a_{2i}b_{i2} & \sum\limits_i a_{2i}b_{i3} \\ \sum\limits_i a_{3i}b_{i1} & \sum\limits_i a_{3i}b_{i2} & \sum\limits_i a_{3i}b_{i3} \end{pmatrix} =$$

$$= \underbrace{\begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} \end{pmatrix}}_{M_1} + \underbrace{\begin{pmatrix} a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} \end{pmatrix}}_{M_2} +$$

$$+ \underbrace{\begin{pmatrix} a_{13}b_{31} & a_{13}b_{32} & a_{13}b_{33} \\ a_{23}b_{31} & a_{23}b_{32} & a_{23}b_{33} \\ a_{33}b_{31} & a_{33}b_{32} & a_{33}b_{33} \end{pmatrix}}_{M_3} \tag{8}$$

So $\mathbf{M}_1$ contains values from the first column of $\mathbf{A}$ and the first row from $\mathbf{B}$, instead of the first row from $\mathbf{A}$ and first column from $\mathbf{B}$. $\mathbf{M}_2$ is using the encond column and row from $\mathbf{A}$ and $\mathbf{B}$, respectively, and $\mathbf{M}_3$ is using the third. Bz reading matrices $\mathbf{A}$ and $\mathbf{B}$ in this way the multiplication can be done in three stages instead of nine.

If the device has $2n$ inputs, the information received is enough for n^2 arithmetic operations instead of only n operations. In other words the information processed grows by the square of the number of inputs instead of linearly, an data-flow is reduced dramatically.

4.2 Integer and powers-of-two weights

The interest in using integer weights stems from the fact that integer multipliers can be implemented more efficiency than floating-point ones. There are also special learning algorithm [11, 12] which use powers-of-two integers as weights. The advantage of powers-of-two integer weight learning algorithm is that the required multiplications in an neural network can be reduced to a series of shift operations. Since this learning algorithm uses the classical BP algorithm with real arithmetics (the first step of adaptation), it is not feasible in stand-alone device.

The fact that despite continuing advantages in digital technology (concrete FPGAs) is still impractical to implement neural networks on FPGAs with floating-point precision weights showed Nichols at al. in his work [15], where they implemented on-line adaptation on the chip (direct implementation of BP).

4.3 Linearly approximated functions

The solution described by Skrbek in [17] is based on linearly approximated functions in combination with the shift operation. It is based on simple functions 2^x, $\log_2 x$ and *sigmoidal function*.

Simple functions The linearly approximated 2^x function can be defined by the expression

$$\mathrm{EXP}_2(x) = 2^{\mathrm{int}(x)}(1 + \mathrm{frac}(x)), \tag{9}$$

where $n = \mathrm{int}(x)$ is integral part of number x and $\mathrm{frac}(x)$ is fractional part of number x. The shift operation calculates 2^n, where $n \in N$. The linear approximation is used in the range of $\langle 2^n, 2^{n+1}\rangle$. The integral part of x assesses the number of shift-left operations for the $(1 + \mathrm{frac}(x))$ expression.

The linearly approximated $\log_2 x$ function can be expressed by

$$\mathrm{LOG}_2(x) = \mathrm{int}(\log_2(x)) + \frac{x}{2^{\mathrm{int}(\log_2(x))}} - 1. \tag{10}$$

For easy implementation, the equation is valid for $x \in \langle 1, \infty)$ and for $x \in \langle 0, 1)$ is generated zero.

The sigmoid or tanh functions are often used an transfer function of a neuron. Both functions are based on e^x which is very complex to calculate. Instead, we use the new function [17] based on the power of two:

$$s(x) = \mathrm{sgn}(x) \left(1 - \frac{1}{2^{|x|}}\right). \tag{11}$$

its linearly approximated version is:

$$S(x) = \mathrm{sgn}(x) \left(\frac{1}{2^{\mathrm{int}(|x|)}} \left(\frac{\mathrm{frac}(|\ x\ |)}{2} - 1\right) + 1\right), \tag{12}$$

and range of this function is $(-1, 1)$.
The small error caused by linear approximation is obvious (see figure 4.3).

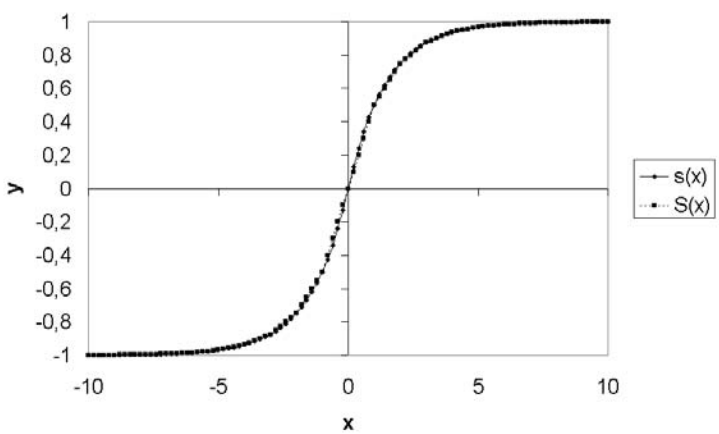

Fig. 2. Graph of the transfer functions $s(x)$ and $S(x)$.

Complex function Complex function are a composition of simple functions. Some complex functions require addtitional add operation to accomplish them. From the set of complex function we describe the most important one: multiplication.

Multiplication is defined by

$$x * y = \mathrm{sgn}(x)\mathrm{sgn}(y)\frac{\mathrm{EXP}_2(\mathrm{LOG}_2(2^n \mid x \mid) + \mathrm{LOG}_2(2^n \mid y \mid)}{2^{2n}}, \qquad (13)$$

where n is the width of the bit grid on which calculations are performed. The arguments x, y and the result are in the range $(-1, 1)$. The sign of the result is evaluated separately.

5 Conclusions

About problems mentioned at the end of section 3: We selected each fifth pattern. This approach revealed itself successful and better than manual pattern selection. The strategies of pattern selection will be expanded in our further work.

The convergention speed during the adaptation phase can be accelerated by other suitable selection strategy (see above) and by hardware implementation. Efficient hardware implementation of neural networks is subject of research at VSB-Technical University of Ostrava.

References

1. Baeza-Yates R., Ribeiro-Neto B.: Modern Information Retrieval. Addison Wesley, New York, 1999.
2. Belohlavek R.: Representation of concept lattices by bidirectional associative memories, research report, Neural Computation 12,10(2000), 2279-2290
3. Berry M. W (Ed.): Survey of Text Mining: Clustering Classification, and Retrieval. Springer Verlag 2003.
4. Dvorský J., Martinovič J., Snášel V.: Query Expansion and Evolution of Topic in Information Retrieval Systems, DATESO 2004, ISBN: 80-248-0457-3.
5. Ganter B., Wille R.: Formal Concept Analysis. Springer-Verlag, Berlin, Heidelberg, 1999.
6. Havel V., Vlcek K., Mitrych J.: Neural Network Architectures for Image Compression. In IFAC PDS 2004, Nov. 18 - 19, 2004, Cracow, Poland, Gliwice, Poland:Silesian University of Technology, , 2004, 389-394, 83-908409-8-7
7. Hecht-Nielsen, R.: Neurocomputing, Addison Wesley, 1991.
8. Hidvegi A.: Implementation of Neural Networks in FPGAs. Dept. of Physics, Stockholm University, 2002.
9. Kiu Ch-Ch, Lee Ch-S.: Discovering Ontological Semantics using FCA and SOM. PROCEEDINGS of M2USIC, Putrajaya, Malaysia 2004.
10. Lendaris G.G (1989), "Experiment on Implementing the Concept-Type Lattice in a Neural Network," Proceedings of AAAI Fourth Annual Workshop (AAAI-89/IJCAI-89) on Conceptual Graphs, AAAI, Menlo Park, CA 94025, July.
11. Marchesi M. et al.: Design of Multi-Layer neural Networks with Power-of-Two Weights. Proceedings of ISCAS-90, IEEE International Symposium on Circuit and Systems, New Orleans (LA-USA), pp. 2951–2954, 1990.
12. Marchesi M. et al.: Fast Neural Networks without Multipliers. IEEE transactions on Neural Networks, 1993. 4(1): pp. 53–62.

13. Nehring K., Puppe, C.: Modelling phylogenetic diversity. Resource and Energy Economics (2002).
14. Nehring K.: A Theory of Diversity. Ecometrica 70 (2002) 1155 1198.
15. Nichols K., Moussa M, Areibi S.: Feasibility of Floating-Point Arithmetic in FPGA based Artificial Neural Networks. in Proceedings of the 15th International Conference on Computer Applications in Industry and Engineering. 2002. San Diego, California.
16. Rojas, R.: Neural Networks, A Systematic Introduction. Springer-Verlag, Berlin 1996
17. Skrbek M.: Fast Neural Network Implementation. Neural Network World 5/99, pp. 375–391, 1999.
18. Zhu, J., Sutton, P.: FPGA Implementations of Neural Networks - A Survey of a Decade of Progress. In: Proc. of the 13th International Conference on Field-Programmable Logic and Applications, LNCS 2778, Springer Verlag, Berlin, pp 1062-1066