Optimizations of the Combinatorial Neural Model

Fábio Ghignatti Beckenkamp
beckenkamp@acm.org
Software Engineering Group, University of Constance
Universitaetsstr. 10
D-78457 Constance, Germany

Miguel Artur Feldens
feldens@inf.ufrgs.br
Computer Science Institute, Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500
Porto Alegre, Brazil

Wolfgang Pree
pree@acm.org
Software Engineering Group, University of Constance
Universitaetsstr. 10
D-78457 Constance, Germany

Abstract

In this paper we present significant optimizations of the so-called Combinatorial Neural Model (CNM). CNM is a hybrid (neural/symbolic) model that has been used in areas such as expert system development and data mining. The paper first explains the CNM architecture and goes on to present CNM optimizations together with empiric results. The most important optimization aims at taming combinatorial explosion, which is the main problem inherent to this model.

1. Introduction

The Combinatorial Neural Model [9, 10] has been explored and developed during the past few years. Experiments with this model have demonstrated that it is well suited for classification problems, with impressive good results in terms of accuracy [5, 6]. Leão and Reátegui [7] developed an expert system, called Hycones that is based on the CNM. Hycones supports the application of CNM to different problem domains. However, Hycones' basic architecture suffers from some flexibility limitations that have been solved in the realm of the JABC project (Java Artificial neural networks Business Components). A generic and flexible architecture constitutes an important goal right from the beginning of the development of this project. The details can be seen in [3, 12]. The central idea of the JABC project is to incorporate object oriented technology to the architecture in order to have more flexibility and reusability [13]. As the basic neural model of this project is the CNM, many implementation aspects naturally arose which improve the CNM model. In the course of this paper, we show the implementation aspects that are proving to extend the CNM performance and usability.

2. The Combinatorial Neural Model (CNM)

The CNM integrates in one straight-forward architecture symbolic and non-symbolic knowledge. This model has characteristics that are desirable in a classification system:

- Simplicity of neural learning - due to a neural network's generalization capacity.
- Explanation capacity – the model can map neural network's knowledge into a symbolic representation.
- High-speed training - only one pass over the training examples is required.
- Immunity against some common neural network pitfalls – i.e. local optima, plateau, etc.
- Incremental learning possibility - previously learned knowledge can be improved with new cases.
- Flexible uncertainty handling – it can accept fuzzy inputs, probabilistic inputs, etc, as inputs fall into the interval [0, 1].

The CNM includes mapping of previous knowledge to the neural network, training algorithms, and pruning criteria, in order to extract only significant pieces of knowledge. The CNM is a 1-hidden layer, feed-forward network. It has particular characteristics in the way the topology is constructed, in neurons, in the links between neurons, and in its training algorithm.

The domain knowledge is mapped to the network through evidences and hypotheses. One evidence may have many distinct values that must be evaluated separately by the neural network, called findings. The input layer represents the defined set of findings (also called *literals*). An example: if an evidence age is modeled, it probably has findings that can be modeled as fuzzy intervals. The problem domain expert defines fuzzy sets for different ages (e.g. teen, youth, adult, senior). Each fuzzy set then corresponds to a finding and, as consequence, to a CNM input neuron. In the CNM, each input value is in the [0,1] interval, indicating the pertinence of the training example to a certain concept, or the degree of confidence.

The intermediate (combinatorial) layer is automatically generated. A neuron is added to this layer for each possible combination of evidences, from order 1 to a maximum order, given by the user.

The output layer corresponds to the possible classes (hypothesis) to which an example could belong. Combinatorial neurons behave as conjunctions of findings that lead to a certain class. For that reason, combinatorial neurons propagate input values according to a fuzzy AND operator, taking as its output the minimum value received by the inputs. Output neurons group the classification hypothesis, implementing a fuzzy OR operator, propagating the maximum value received by its inputs.

The connections between neurons (synapses) have weights and also pairs of accumulators for punishment and reward. Before the training process, in absence of previous knowledge, all weights are set to one and all accumulators to zero. During the training, as each example is presented and propagated, all links that led to the proper classification have their reward accumulators incremented through Backpropagation. Similarly, misclassifications increment the punishment accumulators of the path that led to wrong outputs. Note that weights remain unchanged during the training process, only accumulators are incremented.

The training process is generally done in one pass over the training examples. At the end of this sequential pass, accumulators are used to compute new weights. Based on the CNM topology, symbolic knowledge can be easily extracted, and the new weights can be used to compute the confidence of each piece of knowledge.

3. Optimizations

The main limitation of CNM is the possibility of combinatorial explosion, since the intermediate layer grows exponentially. The combinatorial explosion problem is critical because of memory and processing restrictions that computers have. It is not possible to previously generate all domain problem hypothesis (represented by CNM combinatorial neurons) and subsequently evaluate which one must remain or not. Because of this restriction, until now the CNM model is applied to few areas with maximum combination order of 3. Some effort has been done in trying to avoid these restrictions as using genetic algorithms to increase the maximum combination order [4, 11]. In the next section we present our contribution to this problem.

3.1 Separation of Evidences by Hypothesis

The CNM model is essentially based on the knowledge graphs defined by Leão and Rocha [6]. During the knowledge graphs construction, the domain expert defines which evidences and findings have to be considered. He/she can also determine which evidences/findings relate to each problem hypothesis. This means that for some hypothesis, a smaller number of evidences/findings can be considered. As the CNM neural network structure generation is independent for each defined hypothesis, some of them can have its combinatorial explosion reduced. This happens in reality, for example in a credit analysis problem the expert determined that the evidence sex is important for evaluating bad customers but not for evaluating good ones. So, considering a distinct set of relevant findings for each hypothesis may significantly reduce the search space.

3.2 Avoiding nonsense combinations

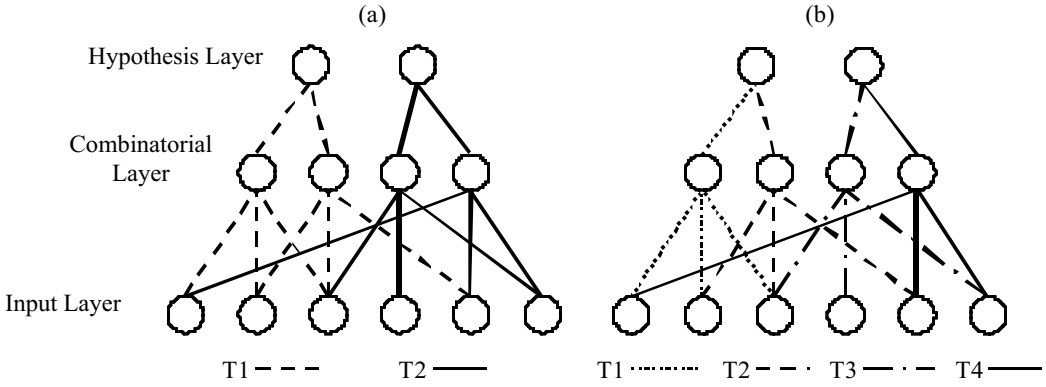
It does not make sense to generate combinations of findings of the same evidence. For example the evidence called “Blood Type” in a medical diagnosis problem. Each blood type is a different finding. For each type one input node is defined. In this case nonsense combinations are those where more than one type is considered because a patient never will have more than one blood type. It is clear and logical, but the original CNM did not consider this kind of situation. This was made probably with the hope of simplifying the implementation algorithm but resulted in compromising the overall system’s efficiency.

3.3 Parallelization and distribution

The object-oriented architecture and the

implementation in Java turns out to facilitate the use of threads in a very intuitive way [3]. Since the combinatorial structure of the whole network is independent of each hypothesis, one thread is created to manage the learning process of each hypothesis. In figure 1(a), T1 represents one thread that controls the combinations of the first hypothesis. T2 is analogous for the second hypothesis. In addition more than one thread can be defined for each hypothesis (Figure 1(b)). The set of generated combinations is divided in as many threads as the user defines. This solution optimizes the use of threads according to the size of the combination layer and the machine capabilities.

Figure 1: Using threads on CNM



The CNM learning and testing can also be distributed. The first tests of this implementation aspect currently being investigated. The basic idea is to make the learning of parts of the neural network in different machines in order to use the several machines available in the lab computer network. The basic technology that has been used is object migration developed with the Voyager library[1]. The criteria in the division of parts of the network to be trained in different machines is based on the same criteria of defining the threads for the network machines processing and memory capacity aspects.

3.4 Optimization on the combination order definition and generation

We have taken a property on which many association rule discovery algorithms [1] are based in order to minimize the threat of the combinatorial explosion: Taking a set of selection criteria, the number of examples which pass such criteria cannot exceed the number of examples selected by any subset of this selection criteria. For example, if patients were selected from a database with the criteria: AGE > 30 AND SEX="FEMALE", the number of patients that will be retrieved cannot be larger than the number of patients that would be selected by one of those criteria taken separately.

Based on such a property, the association rule discovery algorithm Apriori [2] first analyses individual items (which are equivalent to the concept of findings), so that only the ones supported by the examples used in training are combined generating 2-itemsets (combinations of 2 findings). From the 2-itemset combinations, only the ones which are supported by the examples are expanded generating 3-itemset combinations, and so on. With such an inspiration, the CNM algorithm for the topology generation has been optimized, resulting in a major search space reduction, especially for complex applications with a very large number of findings and where high order knowledge has to be discovered. The improved algorithm is shown in Algorithm 1.

Algorithm 1: New algorithm for CNM learning

```
Let Hk be a set of domain problem hypothesis;
Let N be an empty CNM network;
Let Fk be the set of findings that occur within the
set of examples of Hk;
For each order Od from 2 up to N do
Begin
    Let Nkt be an empty temporary CNM network;
    For each hypothesis Hk do
        Add combinatorial neurons to Nkt by combining
Fk with order Od;
    Train network Nkt;
    Prune non-rewarded combinations of network Nkt;
    Add to N network all remaining combinations from
Nkt;
    For each hypothesis Hk do
        Let Fk be the set of findings that appear on
rewarded combinations Nkt;
End;
Prune the N remaining networks by the original CNM
```

[1] Objectspace: http://www.objectspace.com

```
algorithm readjusting the weights;
```

In the first iteration of the main "for" loop, the findings that occur associated to each hypothesis will be considered to generate order 2 combinations. These combinations are stored in a temporary network, which is trained and pruned. By pruning, all combinations that were not validated by the examples will be deleted from the network. This pruning is a simplified version so that only the combinations that were never rewarded during learning (reward accumulator is zero) are pruned and the weights are not changed. After this pruning, the remaining combinations are transferred to the network.

As some parts of the network have been pruned, it is expected that some findings which did not occur in any combination that has been rewarded (which did not occur in the set of examples), will not be relevant in the next algorithm iteration. Since the complexity of combinatorial layer generation is exponential, even a small reduction of the number of findings to be combined has a significant (or measurable) impact on the size of the search space.

After doing the learning loop from the order 2 to the desired order, a final pruning process is applied over the remaining network. This last pruning is the original CNM pruning algorithm, where combinations that received more punishments than rewards are pruned and the weights are modified.

The main advantage of this algorithm is the reduction of memory and time resources for the learning process without compromising accuracy, as no relevant findings are pruned. Furthermore it is possible to generate nets with higher orders than with the original (non-optimized) algorithm. This can be seen in the next section that discusses the performance of the non-optimized and optimized CNM algorithms.

4. Test Results

The domain for this test is that of credit analysis. Real customer data, provided from a company, have been used describing information about customers and what they bought. The task is to classify the customers as either "good" or "bad" ones. The company domain expert (credit analyst) defined the relevant evidences and findings. A set of 13 evidences were identified, e.g. age, sex, order value, type of customer, etc. From these 13 evidences, 32 findings were defined based on finding types such as fuzzy (e.g. age = teen, youth, adult, senior), numeric (e.g. type of customer = 1,2 or 3) or string (e.g. sex = M or F).

To better evaluate the improvements of the optimized algorithm, 3 tests were performed using 3 CNM networks with the following characteristics:

1. A CNM network generated using the normal CNM algorithm [9, 10] that is called here non-optimized network (Table 1). This network contained all combinations for the specified 32 findings from order 2 to order 4.
2. A CNM network generated without the non-sense combinations but still using the non-optimized algorithm (Table 2). This test is important for verifying the size of the remaining network without the non-sense combinations.
3. A CNM network generated by the optimized algorithm (Table 3). The findings are separated by hypothesis, non-sense combinations are eliminated, and the learning is done step by step by eliminating non-relevant findings based on the Algorithm 1.

The tables 1, 2 and 3 show the results of the 3 proposed tests. The table's structure is as follows:

- The column **"Comb. order"** shows the combination order that the CNM network must generate for learning. In case of Tables 1 and 2, the CNM simultaneously generates all combinations from order 2 to the order specified in this column. In case of Table 3, the CNM generates combinations step by step for each combination order from 2 to 4 based on the Algorithm 1.
- The column **"Hypothesis"** shows the two hypotheses considered in the testing problem domain: "good" and "bad". The division into good or bad is important for the evaluation of other columns that separately show the number of combinations for each hypothesis.
- The column **"Number of generated combinations for order N"** shows the number of generated combinations for each order N.
- The **"Remaining rewarded combinations"** column indicates the combinations that were not pruned because they received reward during the learning process (simplified pruning of Algorithm 1).
- The **"Final number of combinations"** column shows combinations which remain after performing the original CNM pruning (the last pruning of the Algorithm 1).
- The **"Findings number"** is the number of findings considered for the combination order N. In these tests the networks start considering all the 32 findings for each hypothesis.
- The column **"Time"** shows the time taken by the learning algorithm to generate the network and to perform the learning and pruning processes. It was used a set of 44 cases (22 good and 22 bad) for the learning. The time is given in milliseconds and minutes.

- The column **"Memo"** shows the amount of memory (in Mbytes) used for doing the learning, i.e. the amount of memory used for allocating the CNM network.

The columns **"Remaining rewarded combinations"** and **"Final number of combinations"** should have the same values for the three algorithm versions. The first test (Table 1) has different values due to remaining combinations among findings of the same fuzzy evidence. Those combinations should be eliminated since two different values of one evidence are forbidden. However, this sometimes occurs with fuzzy evidences because two different fuzzy values can be presented to the network at the same time in a single case (e.g. an age 45 can be considered 0.5 adult and 0.5 senior). Thus there will inevitably be some non-sense combinations remaining at the end of the learning process. This difference is not encountered in the test types two and three because their nonsense combinations are not generated. It is important to realize that these remaining nonsense combinations are not strong enough to be activated, and do not influence the correct performance of the network.

The analysis of the findings which were eliminated during the optimized learning is equally important. In the test type 3 (Table 3), after the learning and pruning of the order 2, it is verified that some findings are eliminated, reducing to 28 findings for the "Good" hypothesis and to 26 findings for the "Bad" hypothesis. The next learning order only generates combinations for those remaining findings, greatly reducing the overall size of the generated network. In this problem domain, the number of findings did not reduce for the orders larger than 2.

The time consumed for the learning process shows a huge difference between the test type 1 and the test type 3 (non-optimized to the optimized). The optimized network spent 68.95% less time than the non-optimized network.

The parallelization of the CNM network also aids the better time performance of the CNM network. Although the 3 tests reported here were performed with the parallelized network, this aspect did not influence the results. The performance of the parallelized network using Java JIT (Just in Time Compilation) is many times better than an early non-parallelized C version of the system; precise tests of this aspect still have to be completed.

The tests also show the very economic use of memory through the optimized learning algorithm. The optimized network (Table 3) used only 43.38% of the memory compared to the non-optimized (Table 1). Because of such memory savings during the learning, it is possible to generate the neural network up to order 5 using the optimization algorithm. It was not possible to generate the order 5 for the non-optimized algorithm because there was not enough memory to support all the combinations

- The column **"Result"** shows the number of correct responses for each hypothesis after testing another data set with 44 cases (22 good and 22 bad).

generated at the same time.

It is possible to verify this economy in memory as well. With the non-optimized algorithm, the combinations (for all orders) are generated and maintained in memory simultaneously. For the Tables 1 and 2 it is necessary to add the orders 2, 3 and 4 of the column **"Number of generated combinations for order"** for both **"good"** and **"bad"** hypotheses. For the optimized algorithm (Table 3), it is only necessary to add the **"Remaining rewarded combinations"** of previous orders, and the column **"Number of generated combinations for order"** for the order in learning process. Considering learning order 4, for the non-optimized network (Table1), the total number of combinations is 82832 while for the optimized network (Table 3) it is 29111, which means a 64.86% reduction. It is important to remember that the number of generated combinations depends on the combination order and on the domain model. Thus, it may change very much from one application domain to another but the optimizations, in terms of the number of generated combinations, will always be relevant.

5. Conclusions

The optimizations presented in this paper have significantly changed the bounds of the generation of the combinatorial layer of the CNM model. In the approach presented here, relevant findings are separated in a subset for each hypothesis (reducing the number of findings to be considered) and nonsense combinations are avoided. A major search space reduction has been achieved, as the generation of combinations is controlled in order to avoid the pre-generation of all possible combinations for a given combination order. A new algorithm with such optimizations was implemented and tested. Also, the parallelization and distribution of the neural network improve the model's performance. Finally, the adequate software architecture makes it possible to consider each detail in the sense of best using the computational resources to make the CNM model applicable.

Acknowledgements

This work has been partially funded by CNPq/Protem (a Brazilian government organization for scientific research), through "SIDI Project" (Institutional Process Number 680059/95.4).

References

[1] Agrawal, R.; Imielinski, T. & Swami, A. 1993. Mining association rules between sets of items in large

databases. In: Proceedings of the *ACM SIGMOD Conference on Management of Data*, 207-216. Washington, DC.

[2] Agrawal, R.; Mannila, H.; Srikant, R.; Toivonen, H. & Verkamo, A. I. 1996. Fast discovery of association rules. *Advances in knowledge discovery and data mining*. Cambridge: AAAI Press/The MIT Press.

[3] Beckenkamp F.G., Pree W. and da Rosa, S.I.V. 1998. Neural Network Framework Components. To appear in Fayad M., Schmidt D.C. and Johnson R. editors, *Object-Oriented Application Framework: Applications and Experiences*, Volume 2, John Wiley.

[4] Denis, F.A.R.M. and Machado, R.J. 1991. *O Modelo Conexionista Evolutivo*. Rio de Janeiro: IBM – Rio Scientific Center (Technical Report CCR - 128).

[5] Feldens, M.A.& Castilho, J. M. V. Data mining with the combinatorial rule model: an application in a health-care relational database. In: XXIII CLEI. Valparaíso, Chile, 1997.

[6] Leão, B. F. and Rocha, A. F. 1990. Proposed Methodology for Knowledge Acquisition: A Study on Congenital Heart Disease Diagnosis. *Methods of Information in Medicine*. 29(1), p. 30-40.

[7] Leão, B.F. and Reátegui, E. 1993. Hycones: a hybrid connectionist expert system. Proceedings of the *Seventeenth Annual Symposium on Computer Applications in Medical Care - SCAMC*, IEEE Computer Society, Maryland.

[8] Leão, B.F. and Reátegui, E. 1993a. A hybrid connectionist expert system to solve classification problems. Proceedings of *Computers in Cardiology*, IEEE Computer, IEEE Computer Society, London.

[9] Machado, R.J. and Rocha, A.F. 1989. *Handling Knowledge in High Order Neural Networks: The Combinatorial Neural Model*. Rio de Janeiro: IBM Rio Scientific Center (Technical Report CCR076).

[10] Machado, R.J. and Rocha, A.F. 1991. The combinatorial neural network: a connectionist model for knowledge based systems. In B. Bouchon-Meunier, R. R. Yager, and L.A. Zadeh, editors, *Uncertainty in Knowledge Bases*. Springer Verlag.

[11] Machado, R.J. and Rocha, A.F. 1992. Evolutive fuzzy neural networks. Proceedings of the *IEEE International Conference on Fuzzy Systems*.

[12] Pree W., Beckenkamp F. and da Rosa S.I.V. 1997. Object-Oriented Design & Implementation of a Flexible Software Architecture for Decision Support Systems. Proceedings of the *9th International Conference on Software Engineering and Knowledge Engineering - SEKE'97*. Madrid.

[13] Pree, W. 1996. *Framework Patterns*. New York City: SIGS Books)