

Testing neural networks using the complete round robin method

Boris Kovalerchuk, Clayton Todd, Dan Henderson

Dept. of Computer Science, Central Washington University, Ellensburg, WA, 98926-7520

borisk@tahoma.cwu.edu toddc.cwu.edu hendersond@cwu.edu

Abstract

The reliability of learning methods depends on testing discovered patterns in the data before they are used for their intended purpose. A novel extension of the round robin testing method is presented in this paper. The novelty includes the mathematical mechanism used based on the theory of monotone Boolean functions and multithreaded parallel processing. This mechanism speeds up required computations. The method has been implemented for backpropagation neural networks and successfully tested for 1024 neural networks by using SP500 data.

1. Approach and Method

One common approach used to test learned regularities is to **divide** the data set D into two parts and use one part for training and another part for validating the discovered patterns. This process is repeated several times and if results are similar to each other than a discovered regularity can be called reliable for data D.

Three major methods for selecting subsets of training data are known as:

1. **Random** selection of subsets,
2. Selection of **disjoint** subsets,
3. Selection of subsets according the

probability distribution.

These methods are sometimes called, respectively, bootstrap aggregation (bagging), cross-validated committees, and boosting [Dietterich, 1997].

Problems of sub-sampling. Different sub-samples of a sample, Tr, can be governed by different regularities. Rejecting and accepting regularities heavily depends on a specific splitting of Tr. This is common for non-stationary financial time series, e.g., bear and bull market trends for different time intervals.

Assume that 70% of Tr is governed by regularity #2 and only 30% by regularity #1. If accidentally, test set A consists of all cases

governed by regularity #1, then regularity #2 found on A' will be rejected although it is true for 70% of the sample Tr.

Therefore, such tests can reflect rather an arbitrary splitting instead of the real strength of regularities on data.

A more comprehensive approach is the **round robin method**. It is designed to eliminate arbitrary splitting by examining several groups of subsets of Tr. If these groups do not cover all possible subsets then round robin approach faces the problem of selecting independent subsets and determining their sizes.

The **complete round robin method** examines **all groups** of subsets of Tr. The obvious drawback with the complete round robin is that there are 2^n possible subsets, where n is the number of groups of objects in the data set. Learning 2^n neural networks is a computational challenge. Below we present an original implementation of the complete round robin method and techniques to speed up required computations along with experimental testing of 1024 neural networks constructed using SP500 data. This method uses the concepts of monotonicity and multithreaded parallel processing for Windows NT. It is applicable to both attribute-based and relational **data mining methods**. However in this paper, the method is illustrated only for neural networks.

Let M be a learning method and D be a set of N objects, represented by m attributes. $D = \{d_i\}, i=1, \dots, N, d_i = (d_{i1}, d_{i2}, \dots, d_{im})$. Method M is applied to data D for knowledge discovery. We assume that data are grouped, for example in a stock time series, the first 250 data objects (days) belong to 1980, the next 250 objects (days) belong to 1981, and so on. To simplify the example, assume we have ten groups (years), $n=10$. Similarly, half years, quarters and other time intervals can be used. There are 210 possible subsets of the data groups. Any of these subsets can be used as training data.

If the data do not represent a time series, then all their complements without constraint can be

used as testing data. For instance, by selecting the odd groups #1, #3, #5, #7 and #9 for training, permits use of the even groups #2, #4, #6, #8 and #10 for testing.

Alternatively, when the data represents a time series, it is reasonable to assume that test data should represent a later time than the training groups. For instance, training data can be the groups #1 to #5 and testing data can be the groups #6 to #10.

For our work, we have taken a third path with completely independent later data C for testing. This allows us to use any of the 210 subsets of data for training.

The hypothesis of monotonicity is used below as a major assumption with the following notation. Let D_1 and D_2 be training data sets and let Perform1 and Perform2 be the performance indicators of learned models using D_1 and D_2 , respectively. We will consider a simplified case of a binary Perform index, where 1 stands for appropriate performance and 0 stands for inappropriate performance. **The hypothesis of monotonicity** (HM) says that:

$$\text{If } D_1 \supseteq D_2 \text{ then Perform1} \geq \text{Perform2.} \quad (1)$$

This hypothesis means that if data set D_1 covers data set D_2 , then performance of method M on D_1 should be better or equal to performance of M on D_2 . The hypothesis assumes that extra data bring more useful information than noise for knowledge discovery. Obviously, the hypothesis is not always true. From 1024 subsets of ten years used to generate the neural network, the algorithm found 683 subsets such that $D_i \supseteq D_j$. We expected that the significant number of them would satisfy the property of monotonicity

$$P(M, D_i) \geq P(M, D_j)$$

Surprisingly, in this experiment monotonicity was observed for all of 683 combinations of years. This is strong evidence for use of monotonicity along with the complete round robin method. In fact, incomplete round robin method assumes some kind of **independence** of training subsets used. Discovered monotonicity shows that independence at least should be tested.

Below we introduce a formal notation, which will allow us to use the concept of monotonicity in the format where available tools from the theory of monotone Boolean functions can be applied.

The **error** Er for data set $D = \{d_i\}$, $i=1, \dots, N$, is the normalized error of all its components d_i :

$$Er = \frac{\sqrt{\sum_{i=1}^N (T(d_i) - J(d_i))^2}}{\sum_{i=1}^N T(d_i)}$$

Here $T(d_i)$ is the actual target value for d_i and $J(d_i)$ is the target value forecast delivered by discovered model J, i.e., the trained neural network in our case.

Performance is measured by the error tolerance (threshold) Q_0 of error Er:

$$\text{Perform} = \begin{cases} 1, & \text{if } Er \leq Q_0 \\ 0, & \text{if } Er > Q_0 \end{cases}$$

Next, we introduce the hypothesis of monotonicity in terms of binary vectors and Perform to be able to use methods from the theory of monotone Boolean functions. Combinations of years are coded as binary vectors $v_i = (v_{i1}, v_{i2}, \dots, v_{i10})$ with 10 components from (0000000000) to (1111111111) with total $2n=1024$ data subsets. In these binary terms, the hypothesis of monotonicity can be rewritten as

$$\text{If } v_i \lesssim v_j \text{ then Perform}_i \geq \text{Perform}_j \quad (2)$$

Here relation $v_i \lesssim v_j$ (“no greater than”) for binary vectors is defined by the ordinary numeric order relation “ \geq ” for the components of these vectors: $v_i \lesssim v_j \Leftrightarrow v_{ik} \geq v_{jk}$ for all $k=1, \dots, 10$.

Note that not every v_i and v_j are comparable with each other by the “ \lesssim ” relation. More formally, we will present Perform as a quality indicator Q:

$$Q(M, D, Q_0) = 1 \Leftrightarrow \text{Perform} = 1, \quad (3)$$

where Q_0 is some performance limit. In this way, we rewrite (2)

$$\text{If } v_i \lesssim v_j \text{ then } Q(M, D_i, Q_0) \geq Q(M, D_j, Q_0) \quad (4)$$

and obtain $Q(M, D, Q_0)$ as a monotone Boolean function of D.

The theory of monotone Boolean functions [Hansel, 1966; Kovalerchuk et al, 1996] is a well-developed theory having mechanisms to speed up computations. This theory is used to speed up the round robin method. Consider a method M and data sets D_1 and D_2 with D_2 contained in D_1 . Informally, according to the hypothesis of monotonicity if it is found that the method M does not perform well on the data D_1 , then it will not perform well on the data D_2 either. Under this assumption, we do not need to test method M on D_2 .

The experiments with SP500 shows that by eliminating these redundant computations it is

possible to run method M 250 times instead of the complete 1024 times. The number of computations depends on a sequence of testing data subsets D_i . To optimize the sequence of testing, Hansel's lemma [Hansel, 1966, Kovalerchuk et al, 1996] from the theory of monotone Boolean functions is applied to so called Hansel's chains of binary vectors. The mathematical monotone Boolean function techniques are presented in [Kovalerchuk et al, 1996].

The general logic of **software** is the following. The set of Hansel chains is generated first and stored. In the exhaustive case we need

1. to generate 1024 subsets using a file preparation program and
2. compute backpropagation for all of them.

Actually, we follow the sequence dictated by the Hansel chains. Therefore, for a given binary vector we produce the corresponding training data and compute backpropagation generating the Perform value. This value is used along with a stored Hansel chains to decide which binary vector (i.e., subset of data) will be used next for learning neural networks. We consider next the implementation of this approach.

2. Multithreaded implementation

The computation process for the round robin method is decomposed into relatively **independent subprocesses**, matching to learning an individual neural network or a group of the neural networks. The multithreaded application in C++ uses both these decompositions, where each subprocess is implemented as an individual thread. The program is designed in such way that it can run in parallel on several processors to further speed up computations. The screen shot of the implementation for the backpropagation neural network are presented in figure 1. The implementation diagram of this project is presented in figure 2.

Threads and the user interface. The processes described at the end of the previous section lend themselves to implementation by threads.

The system creates a new thread for the first vector in each Hansel chain. Each thread accomplishes three **tasks**:

- training file preparation,
- backpropagation using this file, and
- computing the value Perform.

When the thread starts, it paints a small area within the Thread State Watcher on the form to

provide visual feedback to the user. The Perform value is extended, if possible, to the other threads in the chain and the results are printed to a file.



Figure 1. Main screen for complete round robin method software for neural networks

Threads are continually checking the critical section and an event state to see if it is permissible for them to run yet. When it is finally permitted for them to run, a group of them will all run at once, which is forced by the operating system to maintain the highest priority. Since the starting of all the threads for each vector occurs at the same time, the vectors are split up over several computers.

The main program acts as a client, which accesses servers to run threads on different computers. On initialization, the client gathers user information and sends sets of vectors to various servers. Then the servers initialize only threads corresponding to the vectors they have been given. If a server finishes with all its assigned vectors it starts **helping another server** with the vectors it has not yet finished. Instead of having one computer run 250+ threads, six computers can run anywhere from 9 to 90 threads. This provides a speed up of approximately 6 times.

The client is made of two main components: the interface and the monitor. Once settings are made through the interface, the simulation is started. The monitor then controls the process by setting up, updating, and sending work to all the connected servers. This information is communicated in the form of groups of vectors. Once the servers receive their information, they spawn threads to work on the assigned group of vectors. Once the thread has reached a result (Perform value), the information is communicated back to the monitor. The monitor then sends more work to the server if any

remains and updates the display to reflect the work that has been done. Servers that are given work and fail to reply, have their work flagged and once all other results are collected, the flagged work is sent to a different server. If there is still flagged work and no servers are responding, the client itself will execute the remaining work.

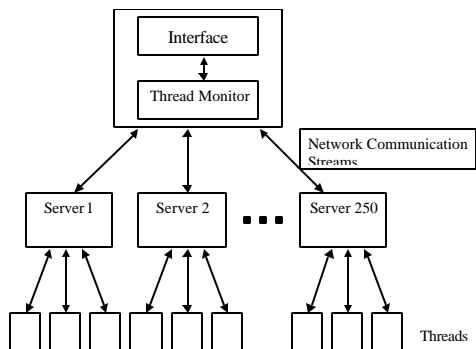


Figure 2. Implementation diagram

3. Experiments with SP500 and Neural Networks

In [Rao, Rao, 1993] the backpropagation neural network producing 0.6% prediction error on the SP500 test data (50 weeks) with 200 weeks (about four years) of training data was presented. Now we consider the question - is this result reliable? In other words, will it be sustained for wider training and testing data? In an effort to answer these questions, we used all available data associated with SP500 from [Rao, Rao, 1993] as training and testing data:

- a) Training data -- all trading weeks from 1980 to 1989 and
- b) Independent testing data -- all trading weeks from 1990-1992.

We generated all 1024 subsets of the ten training years (1980-1989) and computed the corresponding backpropagation neural networks and their Perform values. Table 1 (see the appendix) shows these results. A satisfactory performance is coded as 1 and non-satisfactory performance is coded as 0. Each of the resulting neural networks was tested on the same fixed independent testing data set (1990-1992) with a 3% error tolerance threshold ($Q_0=0.03$), which is higher than 0.6% used for the smaller data set in [Rao, Rao, 1993].

The majority of data subsets (67.05%, 686) satisfied the 3% error tolerance thus demonstrating sound performance of both

training and testing data. Unsound performance was demonstrated by 48 subsets (4.68%). Of those 48 cases, 24 had Perform=1 for training and Perform=0 for testing while the other 24 cases had Perform=0 for training and Perform=1 for testing. Of course testing regularities found on any of those 48 subsets will fail, even if similar regularities were discovered on the 686 other subsets above. Using any of the remaining 289 subsets (28.25%) as training data would lead to the conclusion that there is insufficient data to discover regularities with a 3% error tolerance.

Table 1. Performance of 1024 neural networks

Performance		Number of neural networks	% of neural networks
Training	Testing		
0	0	289	28.25
0	1	24	2.35
1	0	24	2.35
1	1	686	67.05

Therefore, a random choice of data for training from ten-year SP500 data will not produce a regularity in 32.95% of cases, although regularities useful for forecasting do exist.

Table 2 (see appendix) presents a more specific analysis for 9 nested data subsets of the possible 1023 subsets (the trivial empty case (000000000) is excluded from consideration). Suppose we begin the nested sequence with a single year (1986). Not surprisingly, it turns out that this single year's data is too small to train a neural network to a 3% error tolerance. For the next two subsets, the years 1988 and 1989 were added, respectively with the same negative result. However, when a fourth subset was added to the data, 1985, the error moved below the 3% threshold. The additional subsets with five or more years of data also satisfy the error criteria. This, of course, is not surprising as we expect the monotonicity hypothesis will hold.

Similar computations made for all other 1023 combinations of years have shown that only a few combinations of four years satisfy the 3% error tolerance, but practically all five-year combinations satisfy the 3% threshold. In addition, all combinations of over five years satisfy this threshold.

Let us return to the results of Rao and Rao [1993] about the 0.6% error for 200 weeks (about four years) from the 10-year training data. In general, we see that four-year training data sets produce marginally reliable forecasts. For instance, the four years (1980, 1981, 1986, and 1987) corresponding to the binary vector

(1100001100) do not satisfy the 3% error tolerance when used as training data. A neural network trained on them failed when it was tested on 1990-1992 years.

Figure 3 presents further analyses of the performance of the same 1023 data subsets, but with three different levels of error tolerance, Q: 3.5%, 3.0% and 2.0%. The number of neural networks with sound performance goes down from 81.82% to 11.24% by moving error tolerance from 3.5% to 2%. Therefore, neural networks with 3.5% error tolerance are much more reliable than networks with 2.0 % error tolerance. A random choice of training data for 2% error tolerance will more often reject the training data as insufficient. However, this standard approach does not even allow us to know how unreliable the result is without running the complete 1024 subsets in complete round robin method.

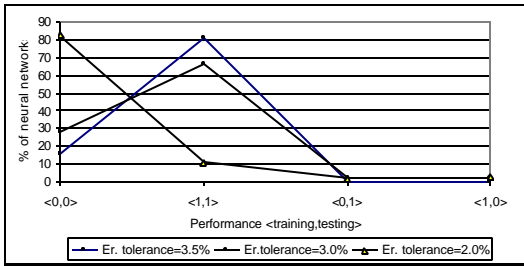


Figure 3. Performance of neural networks.

Running a complete round robin is a computational challenge. Use of monotonicity with 1023 threads decreased average runtime about 3.5 times from 15-20 minutes to 4-6 minutes in order to train 1023 Neural Networks in the case of mixed 1s and 0s for output. Different error tolerance values can change output and runtime. For instance, we may get extreme cases with all 1's or all 0's as outputs. In

addition, a significant amount of time is taken for file preparation to train 1023 Neural Networks. The largest share of time for file preparation (41.5%) is taken by files using five years in the data subset.

In addition, backpropagation neural networks were run on another set of SP500 data (training data -- daily SP500 data from 1985-1994, and testing data -- daily SP500 data from 1995-1998). Figure 2 presents a screen shot of the performance of 256 backpropagation neural networks for these data. Error tolerance in this test was chosen to be 3%. More than one-thousand (1007) neural networks out of 1023 total satisfied this error tolerance on both training and testing data. The total runtime was 4.14 minutes using a single processor. In this experiment, monotonicity allowed us to run 256 subsets instead of 1023 subsets.

We would like to express our gratitude to James Schwing and Edward Gellenbeck for providing many valuable comments.

References

- Dietterich, T. G., Machine Learning Research: Four Current Directions AI Magazine. 18 (4), 97-136. 1997.
- Hansel, G. Sur le nombre des fonctions Boolenes monotones den variables, C.R. Acad. Sci. Paris (in French), 262(20):1088-1090, 1966.
- Kovalerchuk, B., Triantaphyllou, E., Despande, A., Vityaev, E., Interactive Learning of Monotone Boolean Function. Information Sciences, Vol. 94, issue 1-4, 1996, 87-118.
- Rao, V.B., Rao, H.V. C++ Neural Networks and Fuzzy Logic, Management Information Source Press, NY, 1993.

Table 2. Backpropagation neural networks performance for different data subsets.

Binary code for set of years	Training years									Performance		
	80	81	82	83	84	85	86	87	88	89	Training	Testing 90-92
0000001000							x				0	0
0000001001							x			x	0	0
0000001011							x		x	x	0	0
0000011011						x	x		x	x	1	1
0000111011					x	x	x		x	x	1	1
0001111011				x	x	x	x		x	x	1	1
0011111011			x	x	x	x	x		x	x	1	1
0111111011		x	x	x	x	x	x		x	x	1	1
1111111011	x	x	x	x	x	x	x		x	x	1	1