An Adaptive Web Cache Access Predictor Using Neural Network

Wen Tian, Ben Choi, and Vir V. Phoha

Computer Science, College of Engineering and Science
Louisiana Tech University, Ruston, LA71272, USA
Wen_tian2@yahoo.com
pro@BenChoi.org
phoha@coes.latech.edu

Abstract. This paper presents a novel approach to successfully predict Web pages that are most likely to be re-accessed in a given period of time. We present the design of an intelligent predictor that can be implemented on a Web server to guide caching strategies. Our approach is adaptive and learns the changing access patterns of pages in a Web site. The core of our predictor is a neural network that uses a back-propagation learning rule. We present results of the application of this predictor on static data using log files; it can be extended to learn the distribution of live Web page access patterns. Our simulations show fast learning, uniformly good prediction, and up to 82% correct prediction for the following six months based on a one-day training data. This long-range prediction accuracy is attributed to the static structure of the test Web site.

1 Introduction

Use of the Internet is rapidly increasing, and the increase in use motivates the increase in services, and vice versa. All indications are that the use of Internet, services offered, and new applications on the Internet will grow exponentially in the future, resulting in an explosive increase in traffic on the Internet. However, the Internet infrastructure has not kept pace with the increase in traffic, resulting in greater latency for retrieving and loading data on the browsers. Thus, Internet users experience slow response times, especially for popular Web sites. Recently, Web caching has emerged as a promising area to reduce latency in retrieval of Web documents.

In this paper, we present methods and the design of an intelligent predictor for effective Web caching to reduce access latency. Our predictor uses back-propagation neural network to improve the performance of Web caching by predicting the most likely re-accessed objects and then keep these objects in the cache. Our simulation results show promise of successfully capturing Web access patterns, which can be used to reduce latency for Internet users.

Wen Tian, Ben Choi, and Vir Phoha, “An Adaptive Web Cache Access Predictor Using Neural Network,” Developments in Applied Artificial Intelligence, IEA/AIE 2002, Lecture Notes in Artificial Intelligence, Vol. 2358, pp. 450-459, 2002.

Our motivation to use neural networks to predict web accesses follows. The distribution of Web page requests is highly nonlinear and show self-similarity in Web page requests. Since, Neural nets are capable of examining many competing hypotheses at the same time and are more robust than statistical techniques when underlying distributions are generated by non-linear process, it is then natural to use neural nets to predict Web page accesses. Another motivation to use Back-propagation weight update rule in our study is that the Web access logs provide the history of accesses, which can be divided into training examples (test data) to train, and test the predicting power of a supervised learning neural net (see Section 3.2 for our strategy to segment the Web log into training, validation, and test data and our design of NN predictor). Thus, we have all the basic ingredients to build a successful neural network predictor.

The salient features of our approach are: (1) Our predictor learns the patterns inherent in the past Web requests to predict the future Web requests; (2) It adapts to the changing nature of Web requests; (3) It is not dependent on the underlying statistical distribution of Web requests; (4) It is robust to noise and isolated requests; and (5) It can be implemented in hardware or in software.

This paper is organized as follows. In the next subsection, we provide the definition of the symbols used in this paper. In Section 2, we briefly review the related work in caching replacement algorithms. In Section 3, we propose an adaptive Web access predictor using back-propagation neural network to predict the most likely re-accessed objects. In Section 4, we present our simulation results. Finally, in Section 5, we provide conclusion and suggest future work.

1.1 Terminology

This section provides definitions of the symbols used in this paper.

α	A threshold value to determine whether the desired value should be 1 or 0.
β	A threshold value to control learning and prediction granularity.
W_T	The training window
N_T	The size of the training window
W_P	The prediction window
N_P	The size of the prediction window
W_B	The backward-looking window in learning phase
W_F	The forward-looking window in learning phase
W_{F2}	The forward-looking window in predicting phase
N_{F2}	The size of the window W_{F2}
$W_{i,j}$	The weight value from node i to node j in the neural model
η	The learning rate for back-propagation weight update rule.
d	The desired output
t	Time step in the neural net training phase

2 Related Work

Numerous cache replacement algorithms have been proposed by Web caching researchers. The most popular algorithms are: LRU (Least Recently Used) [3], LFU (Least Frequently Used) [3], GDS (Greedy Dual-Size) [14], and LFUDA (LFU with Dynamic Aging) [2], and others reported in [1][2][3] [4] [7] [11] [13].

LRU deletes the objects that have not been requested for the longest time. LFU replaces objects with the lowest access counts; however, this algorithm tends to fill the cache up with frequently accessed old objects. LFUDA is a variant of LFU that uses dynamic aging to accommodate shifts in the set of popular objects. The new feature prevents previously popular objects from polluting the cache by adding an age factor to the reference counts when a new object is added to the cache. GDS takes the size and the cost for retrieving objects into account, this algorithm assigns a value, V, to each object in the cache, the V value is set to the cost of retrieving the object from the origin server, divided by its size. When the cache is full, the object with the smallest V value is replaced.

Few of these algorithms use artificial intelligence techniques to predict Web accesses. Our work builds on the work of Foong, Hu and Heisey [5] who use logistic regression to build an adaptive Web cache. The drawbacks of their approach are: extensive computation required to fit logistic curve for small updates; difficulty of learning access patterns, and absence of a scheduler. Instead, we use a neural network [6] [9] [15], which can be implemented in hardware for real time response, and has fast learning ability. We also present a separate scheduler for training and prediction. We introduce our approach in the following section.

3 Our Design Of A Web Access Predictor

In this section, we present our design of a Web access predictor. A schematic of our intelligent predictor is given in Fig. 1. The predictor has two modules: preprocessing module and processing module. Both of these modules can run as background processes in a Web server.

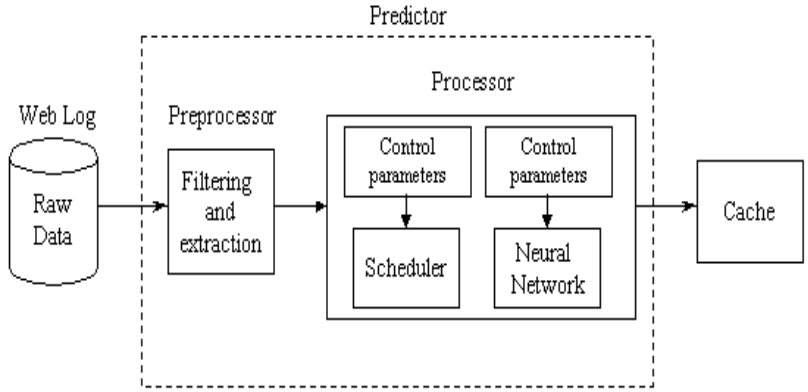

Fig. 1. Schematic of an intelligent predictor

The preprocessor in Fig. 1 handles the filtering and preparation of data in a format to be fed to the processor. The processor consists of a core and a moving window protocol. The design of these components is provided in the follow subsections.

3.1 Preprocessor

For testing our design, we use Web log files from Boston University, spanning the timeframe from November 21, 1994 through May 8, 1995 [8]. The raw data from the log file is in the following form:

<machine name, timestamp, user id, URL, size of the document, object retrieval time in second>

We remove any extraneous data, and transform the useful data into a vector of the form $\{URL, <x_1, x_2, x_3, x_4, x_5>\}$.

X_1 : Type of document
X_2 : Number of previous hits
X_3 : Relative access frequency
X_4 : Size of document (in bytes)
X_5 : Retrieval time(in seconds)

The heuristic to choose value for X_1 is as follows. The values assigned are based on the relative frequency of the type of the files, since there are more image files than HTML files, we assign a higher value to image files and a lower value to HTML files. Since there are few other file types, to penalize learning of other file types, we assign a negative value for all other file types. The following chosen values are somewhat arbitrarily but these chosen values gave us good results in our simulations:

X_1 = 10: HTML files
15: image files
-5: all other files

We also consider the file entries having the same URL but having different file size and different retrieval time as the updated version of previous occurrence of such file. Moreover, if a file entry has size of zero and the retrieval delay is also zero, which means the request was satisfied by the internal cache, then we obtain the file size and the retrieval time from the previous occurrence of the file having the same URL. Next, we can get the document size (X_4) and retrieval time (X_5) directly from the raw data. The process for extracting the value of X_2, and X_3 are provided in Section 3.2.1. These values of X_1, X_2, … X_5 are provided as training vectors for the next stage.

3.2 Processor

The second stage consists of training our neural network using the data obtained from preprocessing. After the network has been trained, we use the network to predict if an object will be re-accessed or not. Fig. 2 shows the design of our predictor using a scheduler and a neural segment.

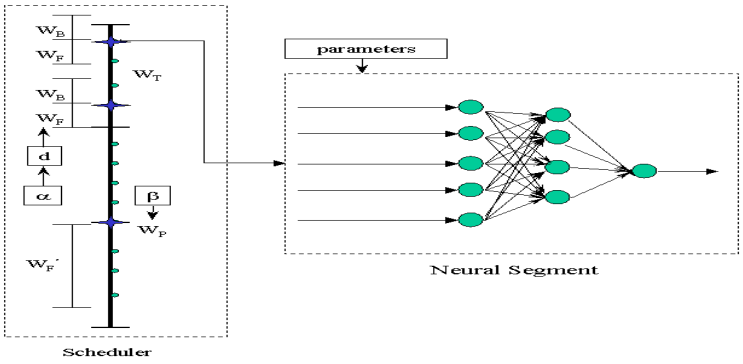

Fig. 2. Design of a neural network architecture to predict cache re-access probability

3.2.1 Training the Predictor Using Back-Propagation Algorithm

As indicated in Fig. 2, the scheduler selects the training data and the prediction windows. The training data was selected using training window W_T, which has size of N_T. We also specify a backward-looking window W_B and a forward-looking window W_F (as was done in [5]). The window W_B and W_F are sliding windows related to current entry. The W_B is the previous log accesses from the current entry while W_F is the following log accesses from the current entry.

Getting Input Vectors

We use the following method to obtain the input vectors for each access log entry.

(1) For retrieving X_1, X_4, X_5, see Section 3.1.
(2) X_2 (previous hit), is equal to how many times an entry has been accessed in window W_B.
(3) X_3 (relative access frequency) is equal to the total number of accesses of a particular URL in window W_T divided by N_T.

After getting the inputs X_1, X_2, X_3, X_4, X_5, we normalize their values to make them in the similar range. Since these variables have different range, such as the size may be 10000 bytes, where as retrieval time may be 2 seconds; so, we scale these entries by dividing each item by a respective constant to scale them to the same range.

Getting Desired Output

The desired output is a binary variable. If the object in W_T has been re-accessed at least α times in W_F, then we assign value one as the desired output, otherwise zero. Higher value of α results in fewer pages to be assigned value one but more accurate prediction, while smaller α value results in more pages to be assigned value one but less accurate prediction.

We modify the standard back-propagation training algorithm [12] [10] [15] to adopt to Web caching. Figure 2 contains our architecture of a feed-forward multi-layer neural network, with five input nodes (corresponding to X_1, X_2, X_3, X_4, X_5), four hidden nodes, and one output node to predict. An outline of back-propagation algorithm follows:

```
function BP-Net(network, training data, α)
returns a network with modified weights
inputs: network, a multilayer network with initialized weight
values between –1 to 1, training data, a set of input/output,
desired output pairs, η is the learning rate
 repeat
   for each e in training data do
     normalize the input
     /* compute the output for this example */
     calculate the input values for the hidden layer
         using sigmoid  function
     calculate the output for the output layer
         using sigmoid function
     /* update the weights leading to the
        output layer */
     /* x_i are input values and y_j s are output values,
        d_j is the desired output */
     W_ij(t+1)←  W_i,j(t)+ η × δ_j × x_i
     where  δ_j = y_j(1- y_j)(d_j- y_j),
            x_i is the input in hidden layer
     /* update the weights in the subsequent layers */
     W_ij(t+1) ←  W_i,j(t)+ η × δ_j × x_i
     where δ_j = x_j(1- x_j)Σδ_kW_jk ,
         k is over all nodes in the layers above node j
     /* calculate the mean square err between
         new weights and old weights */
     Err ← ( W_ij(t+1) - W_i,j(t) )^2
    end
 until network has converged
return network
```

3.2.2 Prediction Stage

After the network has been trained, it was used to predict future access patterns. We selected W_p as the prediction window that consists of N_p entries. The prediction results were compared with the available test log entries. In this stage, we specify a sliding forward-looking window W_{F2} and a threshold β. W_{F2} contains the next N_{F2} access entries related to the current entry. The following pseudo-code shows the method for testing the prediction.

```
boolean correct-prediction(input vector)
   calculate the actual output V with input vector;
   calculate the number of hits H the object has been
      re-accessed in window W_F2;
   /* the predictor predicts the object will
       be re-accessed */
   if ((V >= 0.6) && (H >= •))
     return true;    // correctly predicted
   /* the predictor predicts the object will not be
      re-accessed */
   if ((V < 0.6) && (H < •))
     return true;   // correctly predicted
   else
     return false;
```

4 Simulation Results

In our simulations, we choose $N_T = 500$, $N_P = 5000$, $\alpha = 3$, $N_B = N_F = 50$ and $N_{F2} = 200$. We use log file of one day as the training data. The network converged after 8000 iterations, the mean square error (MSE) value is less than 0.005. Then we use the network to predict access patterns for the next six months. Tables 1 and 2 show the results having β value ranging from 1 to 5. In the tables, learning performance represents the percentage of correct prediction on the data that were part of the training data while prediction performance represents the percentage of correct prediction on data that were not part of the training data.

Table 1. Sample of Simulation Result 1.

Learning data: December 1994, prediction data: January 1995

MSE	LEARNING PERFORMANCE (TESTING)	β	PREDICTION PERFORMANCE
0.00188	80%	1	61%
0.00444	83%	2	67%
0.00417	86%	3	79%
0.00270	85%	4	82%
0.00326	80%	5	79%

Table 2. . Sample of Simulation Result 2

Learning data: December 1994, prediction data: May 1995

MSE	LEARNING PERFORMANCE (TESTING)	β	PREDICTION PERFORMANCE
0.00401	80%	1	61%
0.00331	85%	2	72%
0.00336	82%	3	71%
0.00358	83%	4	75%
0.00042	82%	5	76%

The simulations were done for more than twenty times [16], Tables 1 and 2 are a representative sample of our simulation results. From the tables, we can see that as the β value increases the prediction accuracy increases and it stabilizes for $\beta > 3$. So we choose $\beta = 4$ as the threshold value for this Web site, and we achieve up to 82% percentage of correct prediction (even though the prediction data are six months from the training data). Figure 3 shows the prediction percentage for the next six months for different values of β (we use the training data from one day, then predict the next six months).

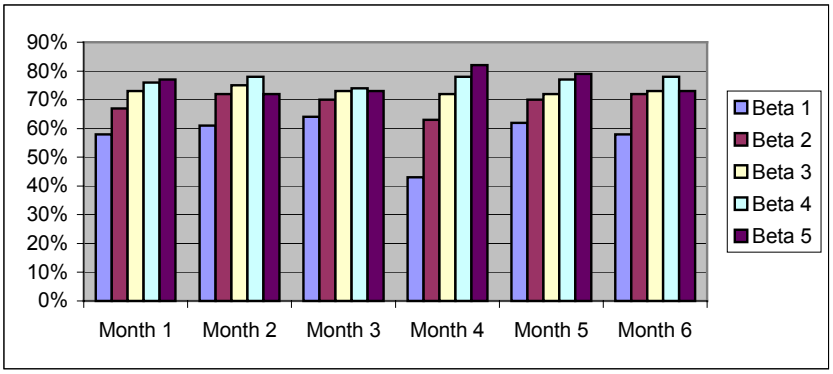

Fig. 3. . Simulation results with different β value

From Fig. 3, we can see that the prediction appears to be uniform (approximately ranging from 60% to 82%, except for month 4 for $\beta = 1$) over a period of six months, we believe this is because the structure of Web site has not changed much over the period of data collected from the Web logs [8].

5 Conclusion and Future Work

In this work, we have successfully presented a novel technique to predict future Web accesses based on history of Web accesses in a Web site. Based on this idea, we have built an intelligent predictor that learns the patterns inherent in the access history of Web pages in a Web site and successfully predicts the future accesses. Our simulations show that the predictor learns the access patterns and we have been able to predict up to 86% accuracy for data on which the network has already been trained and up to 82% accuracy on new and never seen before data. We have also presented heuristics to control the granularity of training data and prediction of the accesses. The success of this technique has opened up many new areas of research. Some of the future areas of further exploration are listed below.

- Develop heuristics to predict the short and long-range period for which current training gives good future prediction. Successful estimate of this time period will help develop strategies to retrain the network.
- Explore other architectures for building the predictor, such as, Self Organizing Feature maps, Adaptive Resonance Theory, Recurrent neural networks, and Radial Basis Function neural network.
- Implement our technique as part of server software, so that the system can be tested in live environment. At present, we have used data collected from log files, which is a static data. We would like to test this technique in a real time and live environment.
- At present, for a significant number of new Web accesses, the method requires retraining. We would like to explore building a dynamic model of a perceptron,

so that only a small part of weights may be changed by adding or deleting nodes. Thus, the training for new patterns would require weight updates for only part of the network. This type of network should learn on the fly without updating the complete set of weights, resulting in no need to completely retrain the network.

Acknowledgements

The access log used in our experiments is available at The Internet Traffic Archive (http://ita.ee.lbl.gov/index.html) sponsored by ACM SIGCOMM. It was collected by the Oceans Research Group (http://cs-www.bu.edu/groups/oceans/Home.html) at Boston University for their work, "Characteristics of WWW Client Traces", which was authored by Carlos A. Cunha, Azer Bestavros, and Mark E. Crovella.

6 References

[1] Ramon Caceres, Fred Douglis, Anja Feldman, Gideon Glass, and Michael Rabinovich, *Web proxy caching*: *the devil is in the details*, 1997~2001 NEC Research Institute.

[2] John Dilley and Martin Arlitt, *Improving Proxy Cache Performance: Analysis of Three Replacement Policies*, IEEE Internet Computing, November-December 1999, pp. 44- 50.

[3] Duane Wessels, *Web Caching*. O'Reilly, 2001.

[4] Li Fan, Pei Cao, and Quinn Jacobson, *Web prefetching between low-bandwidth clients and proxies: potential and performance,* Available at http://www.cs.wisc.edu/~cao/papers/prepush.html (last accessed September 18, 2001.)

[5] Annie P. Foong, Yu-Hen Hu, and Dennis M. Heisey, *Logistic Regression in an Adaptive Web Cache*, IEEE Internet Computing, September-October 1999, pp. 27-36.

[6] Freeman J. and Sakura D. *Neural Networks: Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991.

[7] Dan Foygel and Dennis Strelow, *Reducing Web latency with hierarchical cache-based prefetching,* proceeding of the 2000 international workshop on parallel processing in IEEE 2000.

[8] Internet Traffic Archive, available at http://ita.ee.lbl.gov/html/traces.html (last accessed November 4, 2001.)

[9] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to theory of Neural Computation*, Addison-Wesley, Reading, Mass., 1991.

[10] Jacobs R. *Increased rates of convergence through learning rate adaptation.* Neural Networks 1, 295-307, 1988.

[11] Ludmila Cherkasova, *Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy,* HP laboratories report No. HPL-98-69R1, April, 1998.

[12] Ronald W. Lodewyck and Pi-Sheng Deng, *Experimentation with a back-propagation neural network*, Information and Management 24 (1993) Pp. 1-8.
[13] Evangelos P. Markatos and Catherine E. Chronaki, *A Top-10 approach to prefetching on the Web*. Technical Report 173, ICS-FORTH. Available from http://www.ics.forth.gr/proj/arch-vlsi/www.html (last accessed June 27, 2001.)
[14] N. Young, *On-line caching as cache size varies*, available in the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 241-250, 1991.
[15] R. P. Lippmann, *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine, Vol.4, No.2, Apr.1987, pp. 4-22.
[16] Wen Tian, *Design Of An Adaptive Web Access Predictor Using Neural Network,* MS Report, Computer Science Department, Louisiana Tech University, 2001.