TRAINING AND SIMULATION OF NEURAL NETWORKS FOR WEB PROXY CACHE REPLACEMENT

Jake Cobb and Hala ElAarag
Department of Mathematics and Computer Science
Stetson University
421 N. Woodland Blvd
Deland, FL 32720
wcobb, helaarag@stetson.edu

ABSTRACT
Web proxy caches are widely used to reduce user-perceived latency and link congestion caused by the extremely high volume of web traffic. In this research, neural networks are trained to make proxy cache replacement decisions. The neural networks are trained to classify cacheable objects from real world data sets using information known to be important in web proxy caching, such as frequency, recency and size. The networks are able to obtain correct classification ratios of between .85 and .88 both for data used for training and data not used for training. In simulation, the final neural networks achieve hit rates that are 86.60% of the optimal in the worst case and 100% of the optimal in the best case. Byte-hit rates are 93.36% of the optimal in the worst case and 99.92% of the optimal in the best case.

Keywords

1. INTRODUCTION
Web traffic dominates other contemporary Internet traffic and is continuing to grow in volume. Proxy server caching attempts to reduce user-perceived latency and link congestion by serving local copies of web objects whenever possible. The proxy server aims to serve as many objects from the cache as possible, serve as much data from the cache as possible, or both. Optimizing both is ideal, although not always possible, but many practical algorithms optimize for one over the other.

Web proxy caching is a paging problem. Strategies applied to other paging problems, such as main memory management, have been adapted to address web proxy caching. Web proxy caching has two main factors that must be considered: cache replacement and cache consistency. Cache replacement refers to deciding what should be removed from the cache to make room for new data. Cache consistency refers to ensuring that items in the cache are still the same as items on the original server. This research addresses the former.

In [1], the authors conduct an extensive survey of web cache replacement techniques. Cache replacement strategies are classified into recency-based, frequency-based, recency/frequency-based, function-based and randomized strategies. Recency-based strategies are modeled after Least Recently Used (LRU).
Recency-based strategies center around the idea of temporal locality, meaning that a reference to an item is likely to be followed by additional references. They are simple, quick and somewhat adaptive but many do not handle size information well. Frequency-based strategies are modeled after Least Frequently Used (LFU). These strategies are best for static environments. They are used less often in commercial applications than recency-based strategies because they tend to be more complex. Furthermore, they tend to suffer from cache pollution if an appropriate aging factor is not used. Recency/frequency-based strategies simply consider both recency and frequency in decision-making. Function-based strategies apply a function to candidate items and select for replacement based on that value. The functions have weights or parameters that determine behavior and are thus able to be tuned to specific workloads. Randomized strategies are those that are non-deterministic. Randomized strategies can be fully random, but the vast majority cull choices down to a smaller group and then select an item for replacement at random from the remaining list. Podlipnig et al. [1] list the most commonly considered factors for web proxy cache replacement strategies as a whole as recency, frequency, size, cost to fetch, modification time and expiration time.

In this paper, we present an approach to web proxy cache replacement that uses neural networks for decision making. The framework of this research has appeared in [2]. Neural networks are selected due to their ability to develop internal representations of knowledge, approximate continuous, bounded functions and derive such an approximation by example through supervised learning [3-7].

The rest of the paper is organized as follows. Section 2 presents the related work in web proxy cache replacement and caching applications of neural networks. Section 3 describes Neural Network Proxy Cache Replacement (NNPCR) and enumerates specific methods used in this research. Section 4 reviews the simulation details and results including data preprocessing, metrics, neural network training, and proxy cache simulation. Section 5 presents our conclusions and Section 6 explores areas for future work.

2. RELATED WORK

This section discusses some current web proxy cache replacement algorithms. Next, work using neural networks for caching is reviewed.

2.1 Existing Algorithms

There are a wide variety of web proxy cache replacement algorithms in the literature. According to [1], Pyramidal Selection Scheme (PSS) [8], Greedy Dual Size Frequency (GDSF) [9], Least-Unified Value (LUV) [10] and other algorithms developed from them are considered “good enough” for current web caching needs. The metrics used to determine this designation are discussed later. PSS is a recency-based algorithm. It uses multiple LRU caches and chooses an item for replacement from the selections of the individual LRU lists. GDSF and LUV are both function-based strategies. GDSF extends GD-Size [11] to take account for frequency in addition to size, cost to fetch and an aging factor. LUV uses a sliding window of request times to gather parameters for an undefined function $F(x)$ which is used to calculate the probability $p(i)$ that an
object $i$ will be referenced in the future. This probability is used along with cost to fetch and size information to select an object for replacement.

2.2 Neural Network Caching

Khalid proposed a neural network-based cache replacement scheme [12] called KORA. KORA is based upon earlier work by Pomerene et al. [13] which uses shadow lines. Shadow lines refer to cache lines which are not currently active but have been active in the past. KORA and KORA-2 [14] use neural networks to designate cache lines as shadow lines which then receive preferential treatment. Outside of this distinction, the KORA algorithms use LRU. The KORA algorithms use feed-forward back-propagation neural networks, as does this research. However, KORA addresses traditional cache replacement. Algorithms used in traditional cache replacement form the basis for replacement algorithms used in web caching, but have received extensive modification. Web proxy cache workloads differ from traditional caching due to variable object size and bursty request sequences, amongst other things [1,8-11,15-17].

3. NEURAL NETWORK PROXY CACHE REPLACEMENT

In this section we present our Neural Network Proxy Cache Replacement (NNPCR) technique. The main idea behind NNPCR is to construct and train a neural network to handle web proxy cache replacement decisions. The neural networks used in this research are multi-layer perceptrons (MLP) [5,7]. The weights of the network are adjusted using batch-mode back-propagation with momentum. The sigmoid function shown in equation 1

$$f(u) = \frac{1}{1 + e^{-u}}$$

is used as the activation function. The cross- entropy error function shown in equation 2 is used as the objective function for the neural network.

$$E_{ce} = -\sum_{p} \sum_{o} t_{po} \ln(y_{po}) + (1-t_{po}) \ln(1-y_{po})$$

where,

- $t \in \{0,1\}$ is the target output
- $y$ is the actual output
- $p$ and $o$ are indexed over the training patterns and output nodes, respectively

The neural network has a single output which is interpreted as the probability that the object represented by the inputs is cacheable. NNPCR uses a two hidden layer structure since networks of this class are known to have a universal approximation property. Several sizes and hidden layer configurations were trained in an effort to find the smallest network possible. Networks which are too large often learn the training set well but are unable to generalize [7]. The hidden layers were usually kept at approximately the same size to ease training [7,18]. Section 4.3.1 discusses the effects of these choices.

The cache functions according to the behavior that Podlipnig et al [1] suggest is typical of practical implementations; it caches all objects until it reaches a high mark $H$ and then selects objects for replacement until it reaches a low mark $L$. Network inputs represent recency, frequency and size. Objects are selected for replacement according to the neural network’s output; those with the lowest probability of being cacheable are
replaced first. This approach is a function-based replacement strategy. The network is trained with requests from an hour long segment of a trace file. A second hour long segment is taken from a trace file for a different day in the same week. The second set is used as a verification set. The weights of the network are not adjusted based on the verification set, but the verification patterns are run across the network each epoch to measure how well the network is able to generalize.

NNPCR creates a neural network with small random weights in the range \([-0.6,0.6]\), approximately. The training cycle, or epoch, is repeated until the number of epochs reaches the maximum. Each pattern in the training set is applied to the neural network and the derivatives are back-propagated based on a target output of 0 for un-cacheable requests and 1 for cacheable requests. NNPCR records the error and whether or not the request was classified correctly. A request is classified as cacheable if and only if the neural network output is greater than 0.5. The current set of error derivatives are added to the set of total error derivatives. At the end of each epoch, if the percentage of correctly classified training patterns is the highest seen so far in the training session the network is saved to a file. To ensure good generalization, NNPCR also calculates the correct classification ratio (CCR) the neural network achieves against the verification set. If this ratio is the highest seen so far in the training session, the network is saved to a second file. Finally, the weights are updated based on the set of total error derivatives.

4. SIMULATION

Simulation consists of a simple iterative analysis procedure. Since this research is investigating the web proxy cache replacement problem as it exists independent of specific hardware, low-level cache simulation, such as that performed by DiskSIM [19], is not necessary. The difference in time required to retrieve an object from the disk versus main memory is assumed to be trivial compared to the difference in time required to send an object from the cache versus retrieving and retransmitting a fresh copy of the object. The simulation ran with trace data from IRCache [20]. Unique documents were identified by size and Uniform Resource Identifier (URI).

Rhea et al [15] propose value-based web caching to address concerns about the effects of resource modification and aliasing. Although the point is valid, it is not considered by NNPCR because the actual data transmitted is not available. Furthermore, the trace data is sanitized before being made publicly available so that dynamic information is not available. For example, form data passed by appending the parameters to the URI is removed during sanitation. Finally, such approaches are designed with cache consistency in mind, which is beyond the scope of NNPCR.

4.1 Data Preprocessing

A significant amount of data preprocessing was needed to train the neural network. The IRCache trace files each cover one day and contain tens of thousands of requests. First, we used exhaustive search to convert each log file entry into an entry containing the URI, frequency, size, timestamp and number of future requests for that day. Next, we eliminated internal Squid requests and generated recency values. Recency values were generated by running a cache simulation using the high/low water mark method discussed earlier. The timestamp of the most
recently added or updated request was used as the current time. Lines were written out whenever a request was removed from the cache or caused a cache hit. Items were assigned a rating equal to the number of future requests for that item and replaced starting with the lowest rated. This process created log files with only the desired information and thus made them suitable for training. However, the log files were still excessive in length so one hour of the January 11, 2006 pa.us.ircache.net trace file was selected as the training set. One hour of the January 10, 2006 pa.us.ircache.net trace file was designated as the verification set. pa.us.ircache.net is located in Palo Alto, California. One should note that the information, such as future requests, in each hour long set reflected knowledge of the entire day, not just that particular hour. Finally, the inputs were normalized into the range [-0.9,0.9] to prevent node saturation. This was accomplished by tracking the maximum and minimum values of each property and then applying the following linear normalization function:

\[
x_{\text{norm}} = \frac{x_{\text{actual}} - \min_{\text{data}}}{\max_{\text{data}} - \min_{\text{data}}} \left( \max_{\text{tgt}} - \min_{\text{tgt}} \right) + \min_{\text{tgt}} \tag{3}
\]

4.2 Metrics
Hit-rate and byte-hit-rate are the two most commonly used metrics to evaluate the performance of cache replacement techniques [1,9,16,17]. Podlipnig et al [1] also mention the delay-savings-ratio but claim it is unstable and thus not generally used. Hit-rate refers to the ratio of objects served from the proxy cache versus those retrieved from the original server. This metric targets user perceived delay and availability. Byte-hit-rate is the ratio of number of bytes served from the proxy cache versus retrieved from the original server. Optimizing for this metric reduces the amount of network traffic and thus eases link congestion. Algorithms that favor many small objects in the cache optimize for hit-rate whereas those that prefer fewer large objects optimize for byte-hit-rate. In [1], an algorithm is considered “good enough” for current web proxy caching needs if it performs well for more than one metric. Therefore, both hit rate and byte hit rate are considered in evaluating NNPCR.

For training the network, the most important metric is the correct classification ratio (CCR). We use a combination of sigmoid activation units and the cross-entropy error function, shown in equations 1 and 2 respectively, to interpret network output as the probability that a particular pattern represents a request in the cacheable class. A network output of .75 generates an error signal when the target is 1.0. However, since any network output > 0.5 means the pattern is more likely cacheable than not, this pattern would still be classified correctly. Therefore, correct classification ratio is the criteria used for training purposes while hit rate and byte hit rate are the criteria for performance evaluation once training is complete.

4.3 Performance Evaluation
In order to judge how well the neural network performs, it is compared with LRU and LFU. Additionally, the neural networks performance is compared with a “perfect” function (optimal) which always rates an object with the number of future requests.

The relationship between the correct classification ratio of the training set and that of the verification set is used to measure the neural
network’s ability to generalize. This metric is also applied to sets which were not examined during training to evaluate the practical potential of the neural network.

Several networks were trained with varying structure, learning rates, momentum rates and choice of inputs. Batch mode learning was used for all networks. The majority of the networks converged to the same classification maxima. The maximum correct classification ratio for the training set and verification set was approximately .88. The impact of variations in learning rate (LR) and momentum rate (MR), excluding extreme values, were minimal compared to structural variation. Neural network structure is denoted such that \( a/b/c/d \) means the network has \( a \) inputs, \( b \) nodes in the first hidden layer, \( c \) nodes in the second hidden layer and \( d \) output nodes.

### 4.3.1 Training

Figures 1 and 2 show the correct classification ratio for several different network structures over the first 200 epochs for the training set and verification set, respectively. Figures 3 and 4 show a more detailed look at training behavior by ignoring the brief negative spikes in the CCR values of some networks.

These networks were initialized with different random weights. All the networks used frequency and recency as the first and second inputs, respectively. Neural networks with a hidden layer containing only a single node remained frozen after initialization and were unable to adjust, although lucky initializations sometimes started the networks at fairly high ratios. The smallest structure that was able to learn effectively through
back-propagation was a 2/2/2/1 neural network. In fact, it frequently outperformed larger structures, such as 2/5/5/1, on both the training and verification data sets. This is a very encouraging result because these small networks are fast to train. Adding size as a third input yielded very similar results. For three-input networks, the smallest network able to learn was a 3/2/2/1 structure. The striking similarity between the training and verification graphs indicate the networks are generalizing well. The 2/2/2/1, 2/10/10/1 pair of networks and the 3/2/2/1, 3/5/5/1 pair essentially swapped values for the training set versus the verification set (i.e. one pair’s performance on the training set was equivalent to the other pair’s performance on the verification set). This suggests a tradeoff in performance between sets; it might not be possible to exceed a certain accuracy on a given set without decreasing accuracy on another.

Each neural network was allowed to run 2000 epochs. Many of the networks had jitter in the CCR graph that lent some stochastic search properties to the training once it approached the maxima. As described earlier in the paper, a neural network was saved to file whenever its current set of weights achieved a new highest CCR for either the training or verification set.

4.3.2 Proxy Cache Simulation

Cache simulation was carried out on the entire January 16, 2006 trace file from pa.us.ircache.net. The cache size was set at 0.5GB because larger cache sizes required so few replacements that the problem became too easy to be interesting. Several combinations of high and low mark values were evaluated. The simulation was repeated for the optimal algorithm, LRU, LFU and five different structures of neural network. For each structure, two neural networks were tested: 1) the network saved for best performance on the training set and 2) the network saved for the best performance on the verification set. For a high mark of 0.2 GB, the 3/2/2/1 verification (V) network achieved the highest hit rates of the neural networks and both 2/2/2/1 networks shared the worst hit rates of the networks. Table 1 shows the hit rates of the optimal algorithm, LRU, LFU, the 3/2/2/1 verification network and the 2/2/2/1 networks for a high mark (HM) of 0.2 GB and various low mark (LM) values. Figure 5 illustrates the hit rate performance of all tested algorithms and neural networks for these same parameters.

<table>
<thead>
<tr>
<th>LM (GB):</th>
<th>0.001</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>25.99%</td>
<td>25.99%</td>
<td>26.35%</td>
<td>26.35%</td>
</tr>
<tr>
<td>LRU</td>
<td>23.17%</td>
<td>23.17%</td>
<td>23.69%</td>
<td>24.13%</td>
</tr>
<tr>
<td>LFU</td>
<td>23.60%</td>
<td>23.60%</td>
<td>24.88%</td>
<td>25.11%</td>
</tr>
<tr>
<td>3/2/2/1</td>
<td>24.04%</td>
<td>24.04%</td>
<td>24.90%</td>
<td>25.66%</td>
</tr>
<tr>
<td>2/2/2/1</td>
<td>22.84%</td>
<td>22.84%</td>
<td>22.82%</td>
<td>22.88%</td>
</tr>
</tbody>
</table>

Figure 5. Hit rates for the algorithms and neural networks for HM = 0.2 GB.

Although the 3/2/2/1 verification network performed very well for the hit rate metric, it did the opposite for byte-hit rate. The 2/5/5/1 training (T) network had the highest byte-hit rates of the neural networks, but was the same or slightly worse than LFU. Table 2 shows some selected

<table>
<thead>
<tr>
<th>LM (GB):</th>
<th>0.001</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>25.99%</td>
<td>25.99%</td>
<td>26.35%</td>
<td>26.35%</td>
</tr>
<tr>
<td>LRU</td>
<td>23.17%</td>
<td>23.17%</td>
<td>23.69%</td>
<td>24.13%</td>
</tr>
<tr>
<td>LFU</td>
<td>23.60%</td>
<td>23.60%</td>
<td>24.88%</td>
<td>25.11%</td>
</tr>
<tr>
<td>3/2/2/1</td>
<td>24.04%</td>
<td>24.04%</td>
<td>24.90%</td>
<td>25.66%</td>
</tr>
<tr>
<td>2/2/2/1</td>
<td>22.84%</td>
<td>22.84%</td>
<td>22.82%</td>
<td>22.88%</td>
</tr>
</tbody>
</table>
byte-hit rates and Figure 6 shows all the tested byte-hit rates for a high mark of 0.2 GB.

Table 2. Byte-hit rates for various low mark values and a high mark of 0.2 GB.

<table>
<thead>
<tr>
<th>LM (GB):</th>
<th>0.001</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>25.61%</td>
<td>25.61%</td>
<td>26.20%</td>
<td>26.20%</td>
</tr>
<tr>
<td>LRU</td>
<td>24.00%</td>
<td>24.00%</td>
<td>24.26%</td>
<td>24.65%</td>
</tr>
<tr>
<td>LFU</td>
<td>24.54%</td>
<td>24.54%</td>
<td>24.69%</td>
<td>24.88%</td>
</tr>
<tr>
<td>3/2/2/1</td>
<td>24.09%</td>
<td>24.09%</td>
<td>24.00%</td>
<td>23.92%</td>
</tr>
<tr>
<td>2/5/5/1</td>
<td>24.54%</td>
<td>24.54%</td>
<td>24.46%</td>
<td>24.68%</td>
</tr>
</tbody>
</table>

Figure 6. Byte-hit rates for a high mark of 0.2 GB and various low mark values.

Superior performance for the hit rate metric versus the byte-hit rate metric may be the result of training for classification only and not in terms of a size-related cost.

To further test the effects of cache parameters, the same simulations were run with a high mark of 0.4 GB. Under these conditions, the 2/5/5/1 networks achieved the best hit rates of the neural networks and 2/2/2/1 networks had the worst hit rates. Table 3 compares these networks with the other algorithms and Figure 7 shows the performance of the comparison algorithms and all the neural networks.

Table 3. Algorithm and neural network hit rates for a high mark of 0.4 GB.

<table>
<thead>
<tr>
<th>LM (GB):</th>
<th>0.001</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>24.78%</td>
<td>25.93%</td>
<td>26.35%</td>
<td>26.35%</td>
</tr>
<tr>
<td>LRU</td>
<td>24.68%</td>
<td>24.84%</td>
<td>25.19%</td>
<td>25.36%</td>
</tr>
<tr>
<td>LFU</td>
<td>24.73%</td>
<td>25.05%</td>
<td>25.61%</td>
<td>25.79%</td>
</tr>
<tr>
<td>2/5/5/1</td>
<td>24.78%</td>
<td>24.98%</td>
<td>25.29%</td>
<td>25.41%</td>
</tr>
<tr>
<td>2/2/2/1</td>
<td>24.69%</td>
<td>24.70%</td>
<td>24.78%</td>
<td>24.89%</td>
</tr>
</tbody>
</table>

Figure 7. Algorithm and neural network hit rates for a high mark of 0.4 GB.

For a 0.001 GB low mark, the networks and algorithms perform roughly equivalent. This results from the tight constraint of the small low mark value and the large gap between the high and low mark values. The low mark determines how much is left in the cache after a replacement, so at some point it becomes low enough that not all cacheable requests can be stored at once. This effect is, of course, amplified by poor decisions that leave un-cacheable requests in the cache.

When the gap between the high and low mark values is large, the number of items that must be replaced in a single replacement increases. Furthermore, replacements are carried out less frequently and thus the algorithm or neural network responsible for replacement decisions does not evaluate the status of items in the cache as often. This can be problematic when, for example, many cacheable items are kept during a single replacement sweep but then receive their respective last accesses well before the high mark is reached. Finally, infrequent replacement increases the performance cost of bad replacement decisions because of the additional time selected requests are left untouched in the cache. As the low mark increases, LFU consistently has the high hit rate after the optimal algorithm. The 2/5/5/1 network trails LFU for low marks greater than 0.001 GB, but achieves a better hit rate than LRU for every low mark value. Byte-hit rates for the
neural networks were consistent with the hit rate rankings; the 2/5/5/1 and 2/2/2/1 networks had the highest and lowest byte-hit rates, respectively. Table 4 compares these two networks with the other algorithms and Figure 8 shows byte-hit rates for the algorithms and all the neural networks.

Table 4. Byte-hit rates for the algorithms and neural networks with a high mark of 0.4 GB.

<table>
<thead>
<tr>
<th>LM (GB)</th>
<th>0.001</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>25.00%</td>
<td>26.04%</td>
<td>26.20%</td>
<td>26.20%</td>
</tr>
<tr>
<td>LRU</td>
<td>25.04%</td>
<td>25.21%</td>
<td>25.67%</td>
<td>25.71%</td>
</tr>
<tr>
<td>LFU</td>
<td>25.02%</td>
<td>25.38%</td>
<td>25.58%</td>
<td>25.83%</td>
</tr>
<tr>
<td>2/5/5/1</td>
<td>25.03%</td>
<td>25.57%</td>
<td>25.70%</td>
<td>25.73%</td>
</tr>
<tr>
<td>2/2/2/1</td>
<td>24.98%</td>
<td>24.99%</td>
<td>25.00%</td>
<td>25.06%</td>
</tr>
</tbody>
</table>

Figure 8. Byte-hit rates for the algorithms and neural networks with a high mark of 0.4 GB.

The byte-hit rates were subject to the same constraint as hit rates for a low mark of 0.001 GB. In fact, this was the only case where the optimal algorithm did not achieve the highest metric. However, this is an intuitive result since the optimal algorithm only considers the number of future requests and not the size of a request for replacement decisions. The 2/5/5/1 network performed very well for this metric; it had the highest byte-hit rates after the optimal for 2 of 3 low mark values greater than 0.001. LFU had a higher byte-hit rate when the low mark was 0.1 GB, but the magnitude of the difference between LFU and the 2/5/5/1 network for these parameters was small in comparison to the difference for a low mark of 0.01 GB.

5. CONCLUSION

Web proxy cache replacement is a major component of proxy cache performance. Algorithms have been developed for web proxy caches specifically, as opposed to local caching, because the traffic patterns seen by a web proxy server vary from those seen in local caches on significant characteristics such as variability in object size and locality of references. The respective performances of the replacement algorithms are heavily dependent on metric and workload. Algorithms tend to either have assumptions about workload characteristics built in or include tunable parameters to control which assumption(s) the algorithm favors.

NNPCR is a novel web proxy cache replacement scheme which incorporates a neural network. Supervised learning with real-world data builds workload assumptions into the decision-making process from actual workloads. The neural network approximates a function relating input patterns to a probability that a request will be referenced in the future.

In this paper, we have demonstrated that a properly structured neural network can be efficiently trained to perform web proxy cache replacement from real world data using the basic batch-mode back-propagation with momentum weight update algorithm. Furthermore, we have shown that such a network is able to effectively generalize to other workloads in a similar environment; the neural networks we trained were able to classify both training and verification sets with CCR values in the range of .85 to .88.

We have presented simulation results which suggest this approach is a promising approach to web proxy cache replacement. The neural
networks achieve hits rates of 86.60% and 100% of the optimal in the worst and best cases, respectively. Byte-hit rates range from 93.36% to 99.92% of the optimal in the worst and best cases, respectively. The fact that a variety of structures can be quickly trained for proxy cache replacement with virtually no tweaking of training parameters, such as learning rate, suggests that the capabilities of neural networks are well-suited to the proxy cache replacement problem. Furthermore, since training success for other problems is often dependant on such parameters, neural networks as a class are likely to be capable of even better performance.

6. FUTURE WORK

NNPCR was compared with, aside from the optimal algorithm, LRU and LFU. These algorithms form the basis of other web proxy cache replacement algorithms, but are not the algorithms currently used in practice. Future research which is able to generate neural networks with enhanced performance should compare such networks with more advanced proxy cache replacement algorithms such as PSS, GD-size and LUV.

NNPCR was tested using data from the same proxy cache within a relatively short time frame. The strength of this approach when dealing with different caches and/or greater spans of time needs to be explored. These considerations influence workload characteristics and variability immensely. Workload characteristics undergo significant change over time as technologies and societies develop and change; methods for fast-retraining of the neural network would allow it to adapt to new trends in the data without abandoning the knowledge it previously learned. Finally, the selection of data sets from within the sets made available by IRCache [20] was essentially arbitrary. Although the performance shows that the data was reasonably representative of the workload, statistical methods might be useful for creating training sets that are more representative of general workloads.

7. REFERENCES


