

USING NEURAL NETWORKS FOR
WEB PROXY CACHE REPLACEMENT

by

JAKE COBB

Advisor

HALA ELAARAG

A senior research paper submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science
in the Department of Mathematics and Computer Science
in the College of Arts and Science
at Stetson University
DeLand, Florida

Spring Term
2006

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Hala ElAarag, for all her hard work and guidance on this project and over the course of my undergraduate career. I would also like to thank the rest of the Department of Mathematics and Computer Science faculty, especially Dr. Daniel Plante, for the knowledge and encouragement I have received over the years. Special thanks is extended to Baldur Magnusson for allowing me the benefit of his experience with neural networks.

TABLE OF CONTENTS

TABLE OF CONTENTS.....	iii
ABSTRACT.....	1
1. INTRODUCTION	2
2. BACKGROUND	4
3. RELATED WORK.....	12
4. NEURAL NETWORK PROXY CACHE REPLACEMENT	14
5. SIMULATION.....	18
6. RESULTS	22
7. SUMMARY	39
8. FUTURE WORK.....	42
REFERENCES	44

LIST OF FIGURES

Figure 1. Training set performance over 200 epochs.	23
Figure 2. Verification set performance over 200 epochs.....	23
Figure 3. Training set performance over 200 epochs for $60 < \text{CCR} < 90$	24
Figure 4. Verification set performance over 200 epochs for $60 < \text{CCR} < 90$	24
Figure 5. Hit rates for the algorithms and neural networks for $\text{HM} = 0.2 \text{ GB}$	27
Figure 6. Byte-hit rates for a high mark of 0.2 GB and various low mark values.....	28
Figure 7. Algorithm and neural network hit rates for a high mark of 0.4 GB.	29
Figure 8. Byte-hit rates for the algorithms and neural networks with a high mark of 0.4 GB.	30
Figure 9. Frequency and recency to network output for the 2/2/2/1 network.	32
Figure 10. Frequency and recency to network output for the 2/5/5/1 network.	33
Figure 11. 3/2/2/1 T output for changing recency and size with a frequency of -0.9.....	34
Figure 12. 3/2/2/1 output for changing recency and size with a frequency of -0.6.	35
Figure 13. 3/2/2/1 T output for changing recency and size with a frequency of -0.2.....	35
Figure 14. 3/2/2/1 T output for changing recency and size with a frequency of 0.2.....	36
Figure 15. 3/2/2/1 T output for changing recency and size with a frequency of 0.6.....	36

LIST OF TABLES

Table 1. Hit rates for a high mark of 0.2 GB and several low mark values.	26
Table 2. Byte-hit rates for various low mark values and a high mark of 0.2 GB.	27
Table 3. Algorithm and neural network hit rates for a high mark of 0.4 GB.	28
Table 4. Byte-hit rates for the algorithms and neural networks with a high mark of 0.4 GB.	30

ABSTRACT

Web traffic has continued to increase at a significant rate and is resulting in considerable strain on web servers and bandwidth providers, such as Internet Service Providers (ISP). Proxy caches are often used to reduce this burden. In this research, a novel approach to web proxy cache replacement is developed. Unlike previous approaches, this research utilizes neural networks for 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 and recency. 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.

1. INTRODUCTION

The world wide web accounts for a major portion of contemporary Internet traffic. Popular web sites easily exceed the number of requests, in terms of computational time and bandwidth, that a single server is able to handle. Furthermore, competition amongst web sites makes response time critical. Proxy servers are one method of distributing the load without maintaining multiple exact copies of individual web sites. They are also useful for Internet Service Providers (ISP) who want to make efficient use of limited bandwidth by minimizing unnecessary traffic. The proxy server wants 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, but many practical algorithms optimize for one over the other. There are a number of static algorithms, such as least recently used (LRU) and least frequently used (LFU), which currently address the question of deciding which objects to cache and which objects to remove from the cache in order to accommodate new objects.

Neural networks have gained considerable popularity in recent years. They have been employed in a number of applications, particularly in the area of pattern recognition. Neural networks are able to learn by experience and hold an internal representation of meaning in data. An appropriately structured neural network will be able to generalize the knowledge acquired from training to data that lies outside the training set. This property makes neural networks useful for solving problems that contain uncertainty or have a problem space which is too large for an exhaustive search.

Although a small amount of work that applies neural networks to caching problems exists, until the writing of this paper, there is no work that applies neural networks to web proxy caching specifically. In this research, we present an approach to web proxy cache replacement that uses neural networks for decision making. The rest of the paper is organized as follows. Section 2 presents background information on web proxy cache replacement algorithms and neural networks. Section 3 reviews some related work in web proxy cache replacement and in the application of neural networks to caching problems. Section 4 discusses a novel proxy cache replacement scheme that incorporates neural networks. Section 5 explains the methods and metrics used for simulation and results analysis. Section 6 presents the results of the paper. Section 7 summarizes the previous sections and Section 8 explores areas for future work.

2. BACKGROUND

This section reviews web proxy caching and discusses existing cache replacement schemes. Artificial neural networks will then be reviewed with emphasis on multi-layer feedforward neural networks, which is the organizational method that will be used in this research.

2.1 Web Proxy Caching

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 addressed: 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 will address 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.

2.2 Neural Networks

Neural networks are comprised of many inter-connected simple processing units (neurons) which (when combined) are able to approximate functions based on data sets [2]. Neural networks typically mimic biological neurons by using a set of weights, one for each connection, which is similar to the exciting and inhibiting properties of actual neurons [2-5]. By adjusting these weights such that the network is able to provide correct outputs for most of (ideally all of) the inputs, the network is said to gain knowledge about the problem. This is particularly useful for problems where a definite and/or optimal

algorithm is unknown. Neural networks are also valuable for developing heuristics for problems where the data set is too large for a comprehensive search [3].

A neuron with an associated weight for each connection is known as a McCulloch and Pitts (MCP) neuron [2]. A neural network comprised of MCP neurons is considerably more powerful than a neural network with un-weighted neurons. The weights allow the network to develop its own representation of knowledge [3]. The output of each neuron is determined by applying a function to the sum of every output multiplied by the weight of the associated connection [2-6]. The MCP model does not suggest a specific rule for the translating neuron input to output, so one must be chosen according to the properties of the problem the network is designed to solve.

2.2.1 Firing Rule / Squashing Function

The firing rule determines whether or not the neuron will “fire” based on the inputs. Neuron firing rules are generally linear (proportional output), threshold (binary output), or sigmoid (non-linear proportional output) [2]. Although “firing” is a simple on or off for threshold networks, it is more typically used to mean a rule for determining the amount of output. Neural networks that use non-linear proportional output sometimes refer to the firing rule as the squashing function because they are typically chosen to constrain extreme positive and negative values. The terms firing rule, squashing function and activation function are used interchangeably. Common squashing functions include the sigmoid, tanh and step functions [6]. The sigmoid and tanh functions are approximately linear close to zero but quickly saturate for larger positive or negative numbers. The sigmoid function,

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

is .5 for $u = 0$ and is bounded by 0 and 1. The tanh function,

$$f(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} \quad (2)$$

is 0 for $u = 0$ and is bounded by -1 and 1. Sigmoid and tanh functions both have derivatives which are easy to calculate given the output. For the sigmoid function the derivative is

$$f'(u) = f(u)(1 - f(u)) \quad (3)$$

and for tanh the derivative is

$$f'(u) = 1 - f^2(u) \quad (4)$$

2.2.2 Multilayer Perceptrons

All neural networks have a layer of inputs and a layer of outputs. Neural networks which also feature one or more additional “hidden” layers are called multilayer perceptrons (MLP) [4,6]. In this research, we will employ a feed-forward network. Feed-forward networks are fully connected, meaning all of the neurons in a given layer, proceeding from input to output, are connected to each neuron in the next layer. The output y_i of node i is $f(a_i)$ where f is the activation function and a_i is the activity on node i . The activity is

$$a_i = \sum_{j < i} w_{ij} y_j, \quad (5)$$

where w_{ij} is the weight of the connection to node i from node j . The name is derived from the fact that signals always proceed forward (from input to output) in the network

[2,4,6]. Feedback networks also exist, but are more complicated and less intuitive than feed-forward networks. In a feedback network, neurons may be connected to other neurons in the same layer, a previous layer, or even to itself. These networks, unlike feed-forward networks, fluctuate upon receiving new input until reaching an equilibrium point [2]. A feed-forward MLP with two hidden layers is able to classify regions of any shape [6,7] and approximate any continuous bounded function [6,8].

2.2.3 Supervised Learning

The MLP model described earlier requires the weights to be tuned to the specific problem the network is designed to address. Supervised learning is one method of determining the proper weights. The MLP is trained on inputs where the corresponding target output is known. The weights are then adjusted according to the correctness of the produced output. This process is repeated until the network is able to provide the correct output for each of the training inputs or until a stopping condition is satisfied [2-6]. Since the MLP approximates a function that matches the training data, this approach aims to achieve weights that allow the network to generalize to input/output combinations not included in the training set. In practical applications, the MLP needs two methods: one to measure how closely a training output matches the target output and a second to adjust the weights such that the error level is reduced [2,6].

2.2.4 Objective Function

The objective function, also called the error function, is used to quantify the level of correctness of training output. According to [6], the sum of squared errors is the most

commonly used objective function. However, all objective functions make some assumptions about the distribution of errors and perform accordingly. Some applications require objective functions that lack the benefit of easy differentiability and/or independence found in the most common error functions [6]. The sum of squared errors (SSE) is given by

$$E_{SSE} = \frac{1}{2} \sum_p \sum_o (t_o - y_o)^2 \quad (6)$$

where o is indexed over the output nodes, p is indexed over the training patterns, t_o is the target output and y_o is the actual output. The mean sum of squared error function normalizes SSE for the number of training patterns and is given by

$$E_{MSE} = \frac{1}{P} E_{SSE} \quad (7)$$

where P is the number of training patterns. SSE and MSE both assume a Gaussian error distribution. For classification problems the cross-entropy error function,

$$E_{CE} = - \sum_p \sum_o t_{po} \ln(y_{po}) + (1 - t_{po}) \ln(1 - y_{po}) \quad (8)$$

where $t \in \{0,1\}$. The cross-entropy error function assumes a binomial distribution and outputs and targets are interpreted as a probability or confidence level that the pattern belongs to a certain class.

2.2.5 Back-propagation

Back-propagation is the most commonly used method of adjusting the weights in a MLP. It calculates the partial derivative of the error with respect to the each weight by calculating the rate of change of the error as the activity for each unit changes. The delta

values δ_i are first calculated for the output layer directly based on the target and actual outputs. For SSE, this is

$$\delta_i = f_i'(t_i - y_i) \quad (9)$$

It is not necessary, fortunately, to calculate the derivative of the cross-entropy error function. If the cross-entropy error function is used with sigmoid units, then δ_i for output nodes is simply

$$\delta_i = t_i - y_i \quad (10)$$

The same calculations are then performed for each hidden layer based on information from the layer “in front” of that layer. Once the error derivative of the weights is known, they can be adjusted to reduce the error. Back-propagation is so named because the error derivatives are calculated in the opposite direction of signal propagation. The delta value δ_i of node i for hidden nodes is calculated as

$$\delta_i = f_i' \sum_{k>i} w_{ki} \delta_k, \quad (11)$$

where $i < k$. This process is repeated until the error is less than a threshold determined by the application of the network [2-6].

Reed and Marks [6] point out that back-propagation actually refers to both the derivative calculation and a weight change algorithm. The basic weight update algorithm changes each weight by negative the derivative of the error with respect to the weights multiplied by a small constant η known as the learning rate (LR), as shown in equation 12.

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} \quad (12)$$

Momentum is a common modification to the weight update algorithm which involves adding a fraction of the previous weight change to the current weight change [6].

The back-propagation weight update amount with momentum is

$$\Delta w_{ij}(t) = \eta \frac{\partial E}{\partial w_{ij}}(t) + \alpha \Delta w_{ij}(t-1) \quad (13)$$

where $0 \leq \alpha < 1$ is the momentum rate (MR). Momentum useful for coasting out of a poor local minima in the error surface and traversing flat areas quickly [6].

Two common variations of this algorithm are batch-mode and on-line learning. Batch-mode runs all patterns in a training set. The error derivative with respect to the weight for each pattern is summed to obtain the total error derivative with respect to the weights. All weights are then adjusted accordingly. On-line training, on the other hand, runs a pattern at random from the training set. The weights are updated after each single pattern using the error derivative with respect to the weights for the current training pattern [6].

3. RELATED WORK

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

3.1 Existing Algorithms

There are a wide variety of web proxy cache replacement algorithms in the literature. According to [1], Pyramidal Selection Scheme (PSS) [9], Greedy Dual Size Frequency (GDSF) [10], Least-Unified Value (LUV) [11] 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 [12] 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.

3.2 Neural Network Caching

Khalid proposed a neural network-based cache replacement scheme [13] called KORA. KORA is based upon earlier work by Pomerene et al. [14] 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 [15] 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 will 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 [1,9-12,17-19].

4. 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 multi-layer feed-forward artificial neural network to handle web proxy cache replacement decisions. The weights of the network are adjusted using back-propagation. The sigmoid and cross-entropy error functions are used as the activation function and objective function, respectively, for the neural network. 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 [6]. The hidden layers were usually kept at approximately the same size to ease training [6,16]. The effects of these choices are discussed in section seven.

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. Neural networks are often used to approximate functions from a sample of the data set [6]. 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 the function.

NNPCR creates a neural network with small random weights in the range $[-\frac{1.1}{\sqrt{I}}, \frac{1.1}{\sqrt{I}}]$ where I is the number of inputs. The training cycle, or epoch, is repeated until the number of epochs reaches the maximum. For online training, the order of the training sets is randomized before each epoch. 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. For online training, the weights are adjusted immediately using the current set of error derivatives. For batch training, 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 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, if NNPCR is using batch training the weights are updated based on the set of total error derivatives. The following pseudo-code demonstrates how NNPCR constructs and trains the neural network employed in

proxy cache replacement decisions assuming the previously mentioned high and low mark method is used.

```
neural_network* nnpkr_gen(int hidden_layers, int[] nodes,
                          int update_mode, int max_epochs,
                          trace_set training_set,
                          trace_set verify_set)
{
    neural_net = neural_network(hidden_layers + 2, nodes);
    initialize neural_net weights with small random values;
    iterations = 0;

    /*
    / total_edw is a 3-dimensional array, indexed by layer and
    / sending and receiving nodes, in that order. For example,
    / total_edw[1][2][0] refers to the weighted connection from
    / the third node in the first hidden layer (second actual layer)
    / to the first node in the second hidden layer.
    */

    do
    {
        if(update_mode == ONLINE)
            randomize order of training_set;
        else //update_mode == BATCH
            set all values of total_edw array to 0;

        for(i = 0; i < length(training_set); i++)
        {
            neural_net.forward_prop(training_set[i].inputs);

            if(training_set[i].is_cachable)
                neural_net.back_prop(1.0);
            else
                neural_net.back_prop(0.0);

            edw = neural_net.error_derivatives();

            if(update_mode == ONLINE)
                neural_net.update_weights(edw);
            else //update_mode == BATCH
            {
                for(j = 0; j < edw.depth; j++)
                    for(k = 0; k < edw.width; k++)
                        for(l = 0; l < edw.height; l++)
                            total_edw[j][k][l] += edw[j][k][l];
            }
        }

        if(most training patterns classified so far)
            neural_net.save(best_training_net_file);
    }
}
```

```

    for(i = 0; i < length(verify_set); i++)
        record number correctly classified;

    if(most verification patterns classified so far)
        neural_net.save(best_verify_net_file);

    if(update_mode == BATCH)
        neural_net.update_weights(total_edw);

        iterations++;
} while( iterations < MAX_ITERATIONS );

return &neural_net;
}

```

NNPCR does not dictate exactly when replacement should occur, although training set simulation follows the high/low mark method from [1]. Once the cache is ready to replace one or more objects, NNPCR rates each candidate replacement object by applying each object's characteristics across the inputs of the neural network. NNPCR is flexible about the choice of characteristics but requires the characteristics used in application to match those used in training in terms of both format and semantics. The neural network's output is used as a replacement rating for the object. A lower score represents a better choice for replacement; one or more objects are selected for replacement using these ratings.

5. 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 [20], 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 [21]. Unique documents were identified by size and Uniform Resource Identifier (URI).

Rhea et al [17] propose value-based web caching to address concerns about the effects of resource modification and aliasing. Although the point is valid, it will not be considered by NNPCR because the actual data transmitted will not be 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.

5.1 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 the hour long sets reflected knowledge of the entire day, not just the 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:

$$x_{norm} = \frac{x_{actual} - \min_{data}}{\max_{data} - \min_{data}} (\max_{tgt}) + \min_{tgt} \quad (14)$$

5.2 Metrics

Hit-rate and byte-hit-rate are the two most commonly used metrics to evaluate the performance of cache replacement techniques [1,10,18,19]. Podlipnig et al [1] also mention the delay-savings-ratio but claims 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 will be 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 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 considered complete.

5.3 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.

6. RESULTS

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.

6.1 Training

Figures 1 and 2 show the correct classification ratio (CCR) 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.

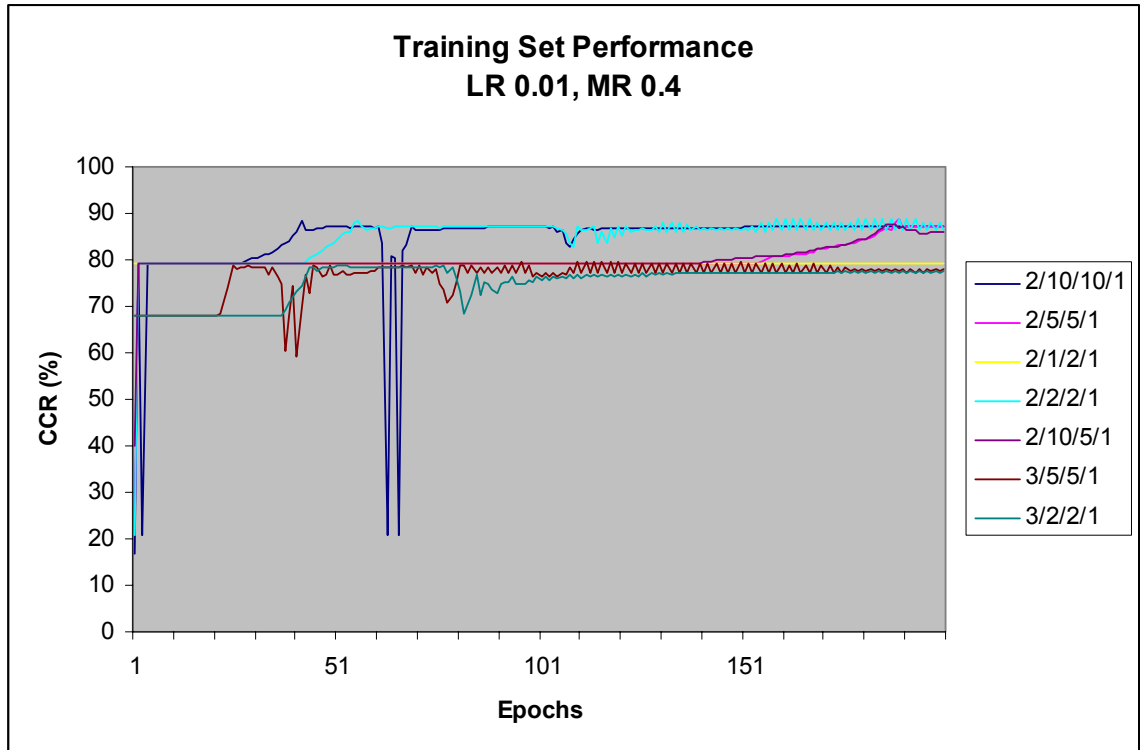


Figure 1. Training set performance over 200 epochs.

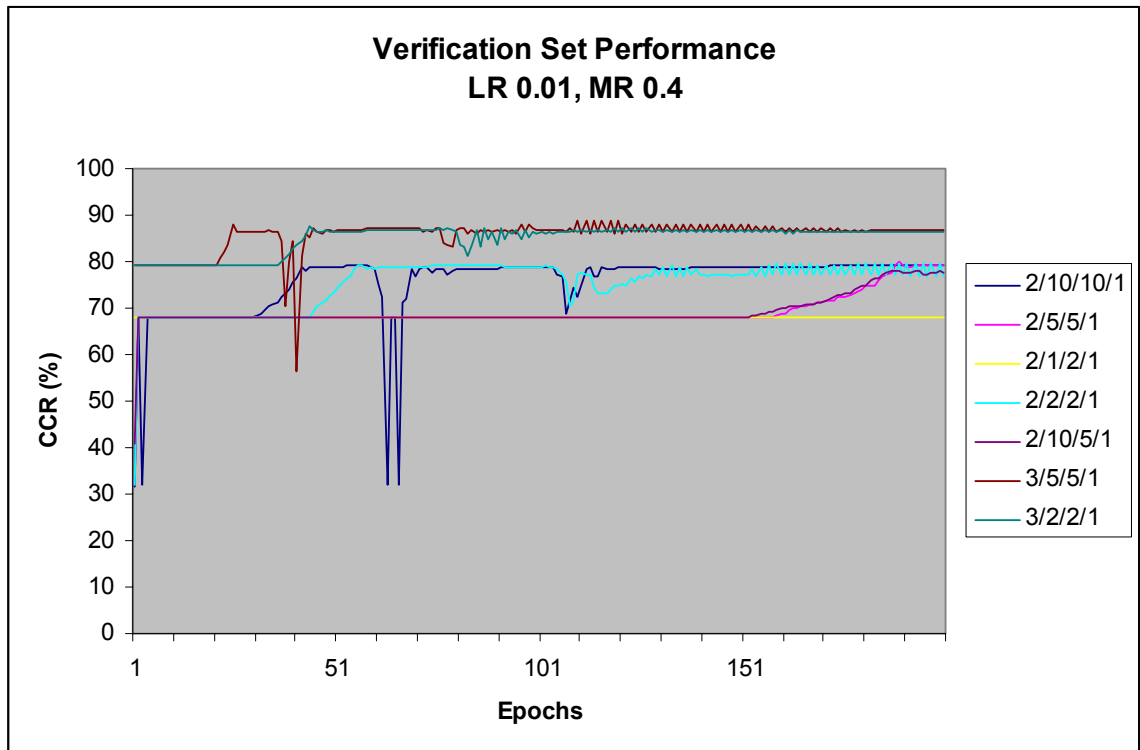


Figure 2. Verification set performance over 200 epochs.

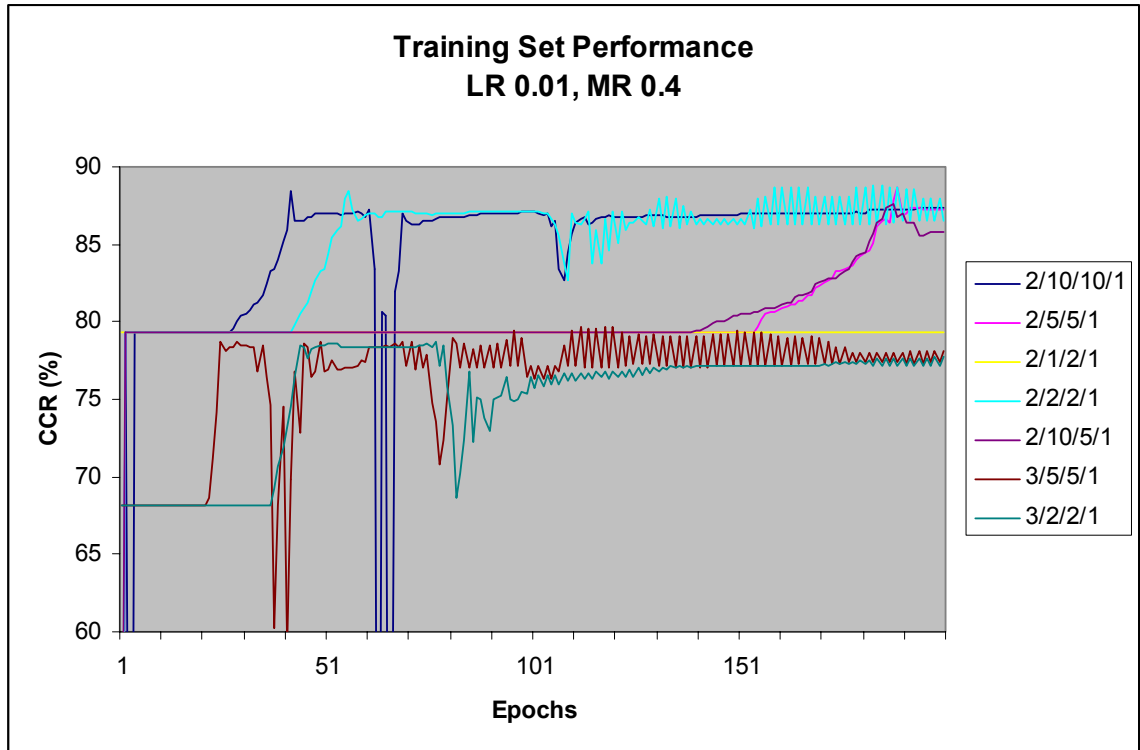


Figure 3. Training set performance over 200 epochs for $60 < CCR < 90$.

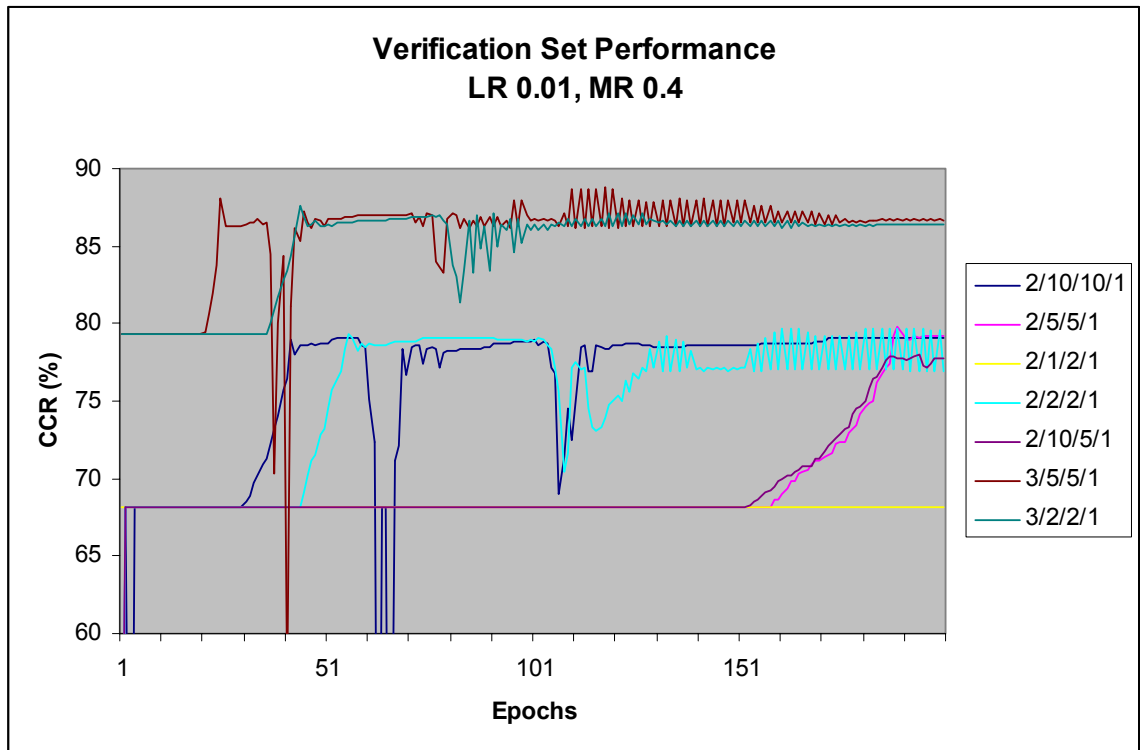


Figure 4. Verification set performance over 200 epochs for $60 < CCR < 90$.

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 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.

6.2 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. 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 1. Hit rates for a high mark of 0.2 GB and several low mark values.

	LM: 0.001	LM: 0.01	LM: 0.05	LM: 0.1
Optimal	25.99%	25.99%	26.35%	26.35%
LRU	23.17%	23.17%	23.69%	24.13%
LFU	23.60%	23.60%	24.88%	25.11%
3/2/2/1	24.04%	24.04%	24.90%	25.66%
2/2/2/1	22.84%	22.84%	22.82%	22.88%

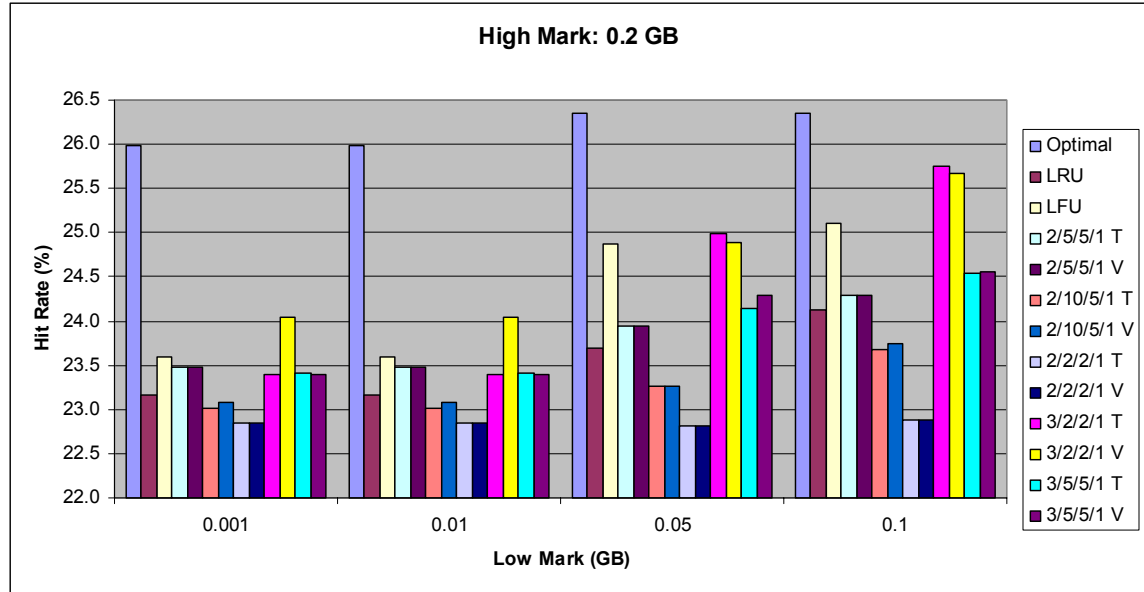


Figure 5. Hit rates for the algorithms and neural networks for HM = 0.2 GB.

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 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.

	LM: 0.001	LM: 0.01	LM: 0.05	LM: 0.1
Optimal	25.61%	25.61%	26.20%	26.20%
LRU	24.00%	24.00%	24.26%	24.65%
LFU	24.54%	24.54%	24.69%	24.88%
3/2/2/1	24.09%	24.09%	24.00%	23.92%
2/5/5/1	24.54%	24.54%	24.46%	24.68%

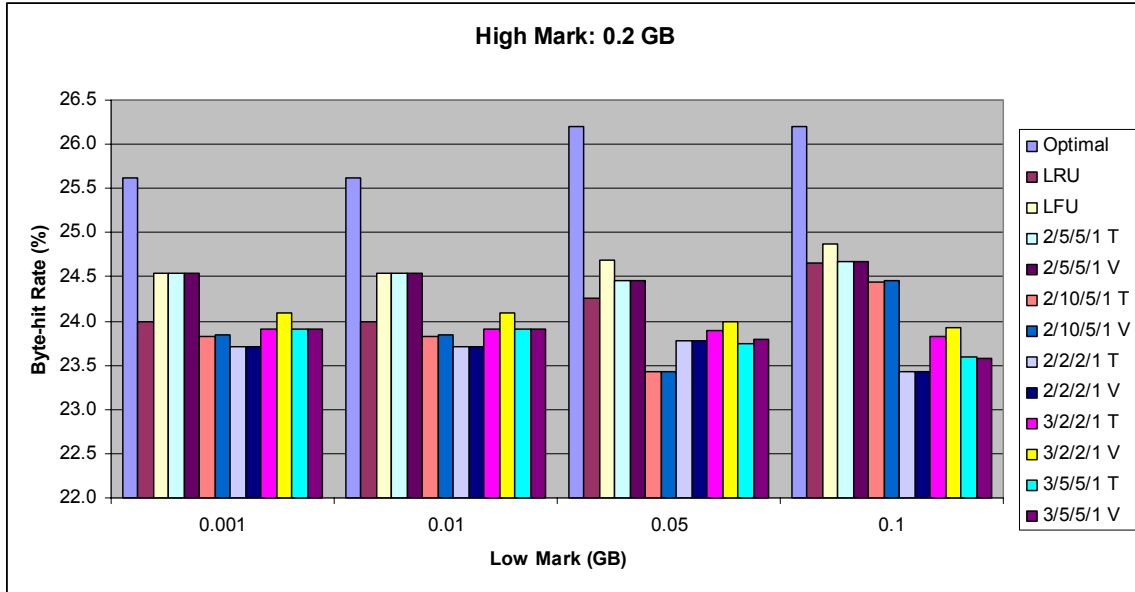


Figure 6. Byte-hit rates for a high mark of 0.2 GB and various low mark values.

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 0.4GB. 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 comparison 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.

	LM: 0.001	LM: 0.01	LM: 0.05	LM: 0.1
Optimal	24.78%	25.93%	26.35%	26.35%
LRU	24.68%	24.84%	25.19%	25.36%
LFU	24.73%	25.05%	25.61%	25.79%
2/5/5/1	24.78%	24.98%	25.29%	25.41%
2/2/2/1	24.69%	24.70%	24.78%	24.89%

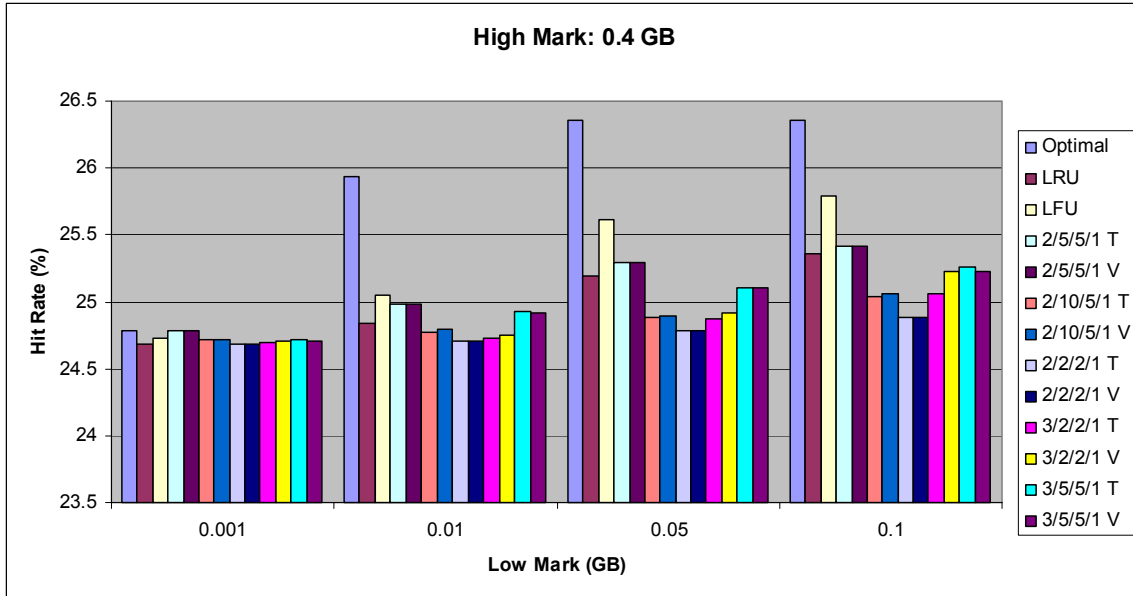


Figure 7. Algorithm and neural network hit rates for a high mark of 0.4 GB.

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.

	LM: 0.001	LM: 0.01	LM: 0.05	LM: 0.1
Optimal	25.00%	26.04%	26.20%	26.20%
LRU	25.04%	25.21%	25.67%	25.71%
LFU	25.02%	25.38%	25.58%	25.83%
2/5/5/1	25.03%	25.57%	25.70%	25.73%
2/2/2/1	24.98%	24.99%	25.00%	25.06%

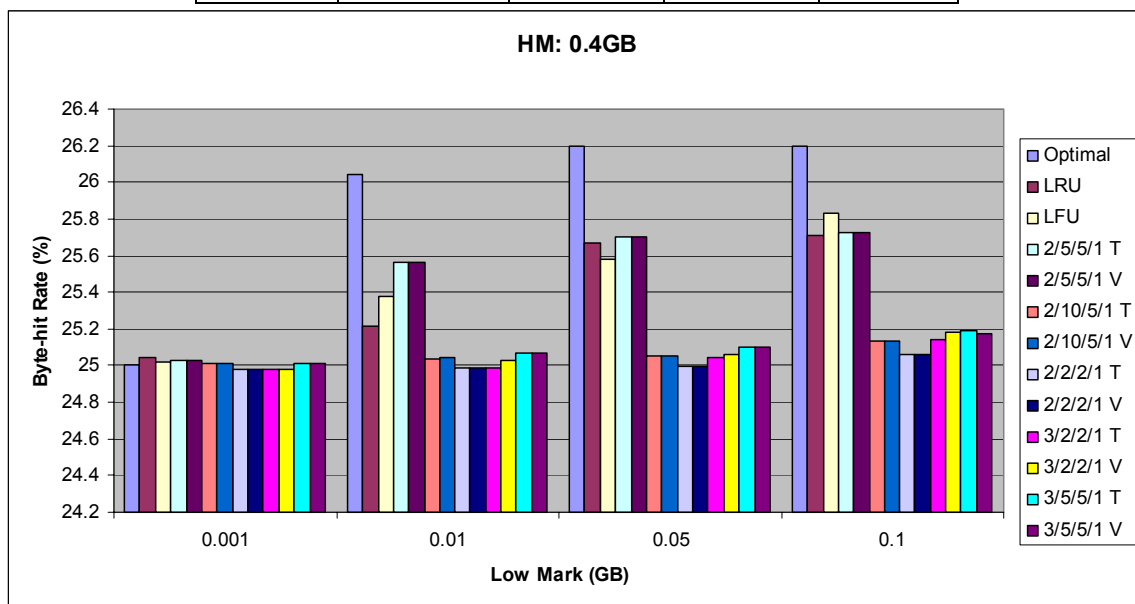


Figure 8. Byte-hit rates for the algorithms and neural networks with a high mark of 0.4 GB.

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.

6.3 Neural Network Input/Output Relationships

The proxy cache simulation results show the neural networks are able to make competitive replacement decisions. Although no neural network fell behind the optimal algorithm by more than 2 percentage points, the local performance variation was significant. Additionally, changing cache parameters had a strong impact on individual neural network performance. In order to better understand the results presented earlier, an attempt was made to determine the characteristics of the individual networks that factor into proxy cache replacement performance.

We first note that the CCR used to measure training and generalization success did not predict the level of success in the simulation. The 2/2/2/1 networks had some of the highest CCR values on the training set, but performed the worst in simulation for all but one case. The 2/5/5/1 networks were relatively close to the 2/2/2/1 networks in CCR for both the training and verification sets, albeit slightly slower to converge, yet had the best overall performance of any of the neural networks. Since neural networks

approximate functions from points of data, there are numerous functions a given network might approximate which are all best-fit for that data. In this application, the function is unknown so the accuracy of a given network depends on how well its internal function matches the unknown function, not just the data points. The rest of this section explores the input-to-output mapping of the neural networks.

Two-input neural networks were trained with normalized frequency and recency values in the range $[-0.9, 0.9]$. The $2/5/5/1$ networks and the $2/2/2/1$ networks were the best and worse neural networks, respectively. However, these networks have very similar structure. Figures 9 and 10 show each network's output for different recency (x-axis) and frequency (series) values.

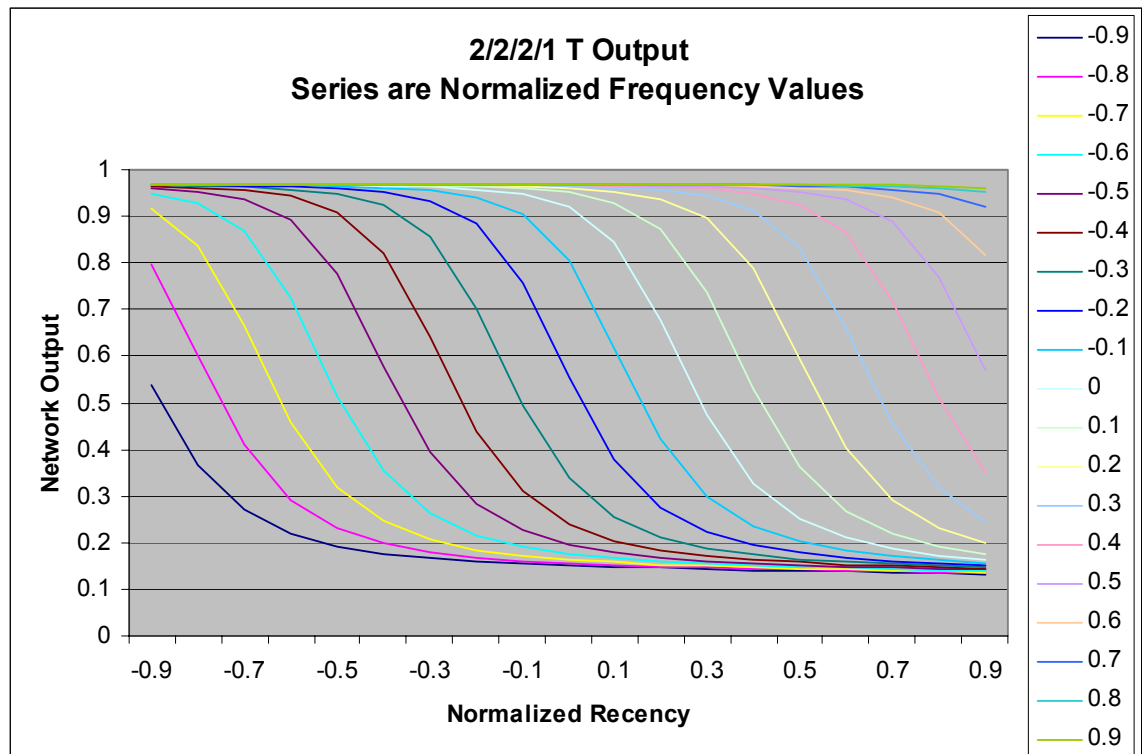


Figure 9. Frequency and recency to network output for the $2/2/2/1$ network.

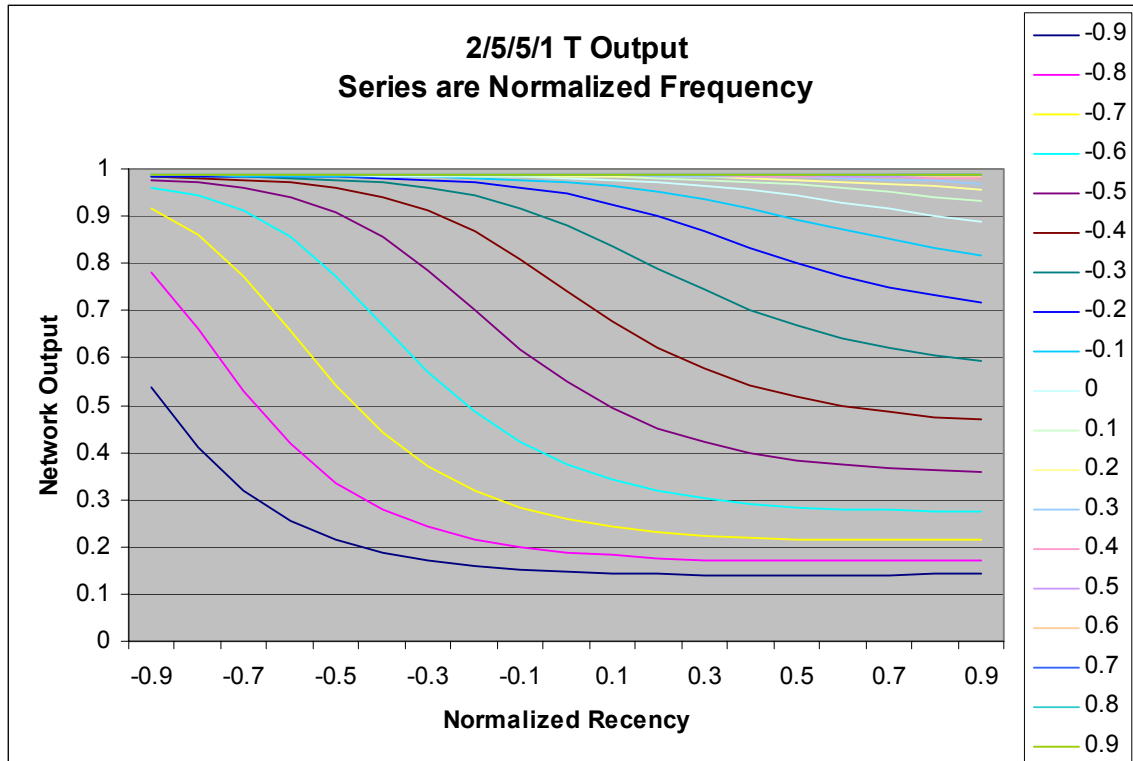


Figure 10. Frequency and recency to network output for the 2/5/5/1 network.

The 2/2/2/1 network outputs resemble “backwards” sigmoid units. Increases in frequency seem to shift the sigmoid further down the recency axis. The steep slope in the middle indicates the network quickly changes its classifications with subtle changes in the input. The 2/5/5/1 network outputs, on the other hand, are much smoother and become even more so as the frequency increases. The 2/5/5/1 outputs give preference to frequency and decrease gradually with increasing recency values (i.e. more time since the last access). Furthermore, as the frequency increases the rate at which increased recency lowers the output is reduced. In other words, the 2/5/5/1 network has a more “intelligent” approach than the 2/2/2/1 network. The 2/2/2/1 network grasps the inverse relationship but models it crudely; for many frequency values the output has a long tail with little output change as recency increases followed by a sharp drop to low output. In contrast, the 2/5/5/1 network’s behavior is similar to LFU with an aging factor.

Three-input networks used a normalized size value for the third input. Usually, the 3/2/2/1 and 3/5/5/1 networks did not stand out during simulation, but the 3/2/2/1 networks, particularly the verification network, achieved hit rates significantly higher than LRU, LFU and the other neural networks for a high mark of 0.2 GB, regardless of the low mark level. . Figures 11 – 17 show the network output for the 3/2/2/1 T network as recency (x-axis) and size (series) change; each figure is for a sequential frequency value from the set $\{-0.9, -0.6, -0.2, 0.2, 0.6\}$.

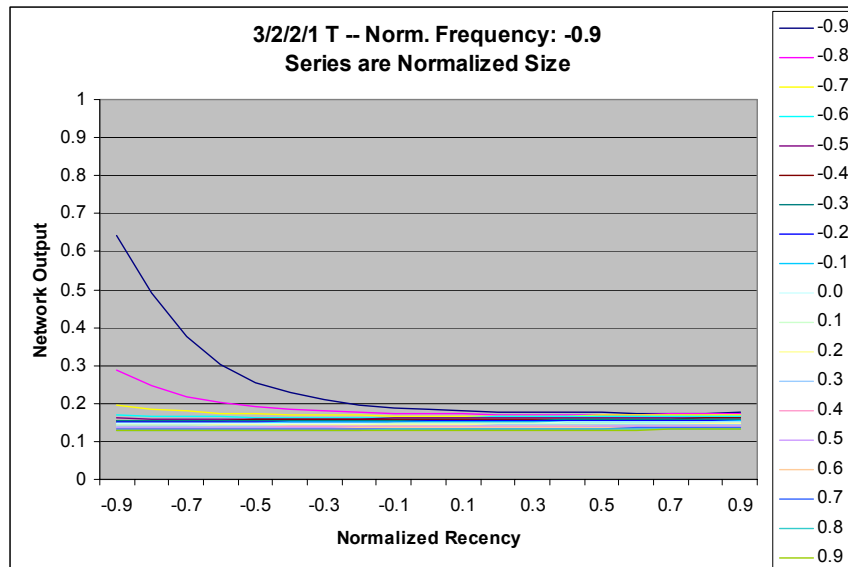


Figure 11. 3/2/2/1 T output for changing recency and size with a frequency of -0.9.

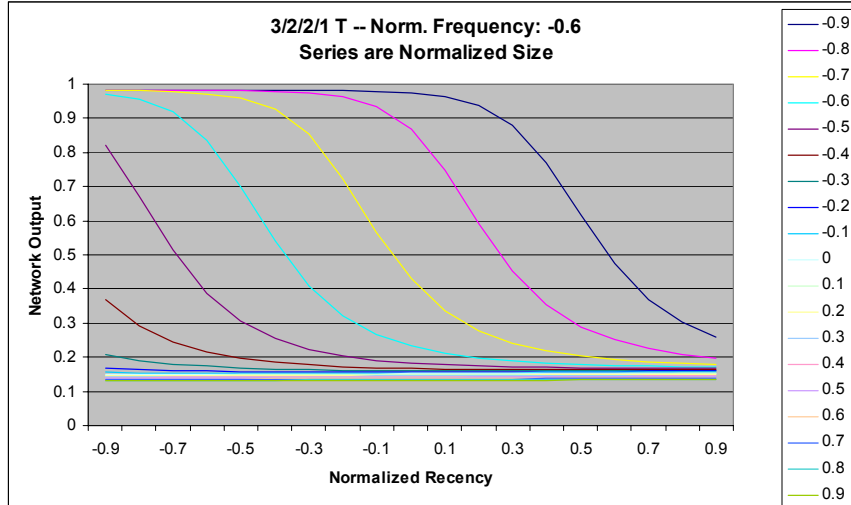


Figure 12. 3/2/2/1 output for changing recency and size with a frequency of -0.6.

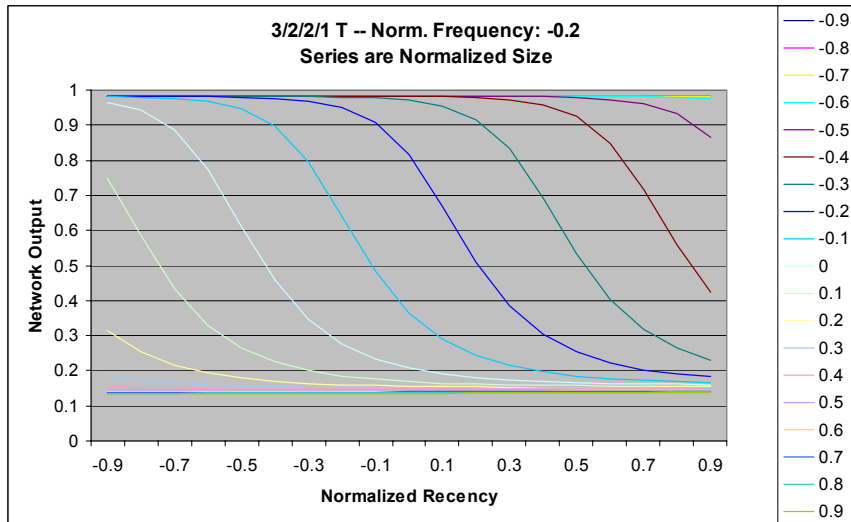


Figure 13. 3/2/2/1 T output for changing recency and size with a frequency of -0.2.

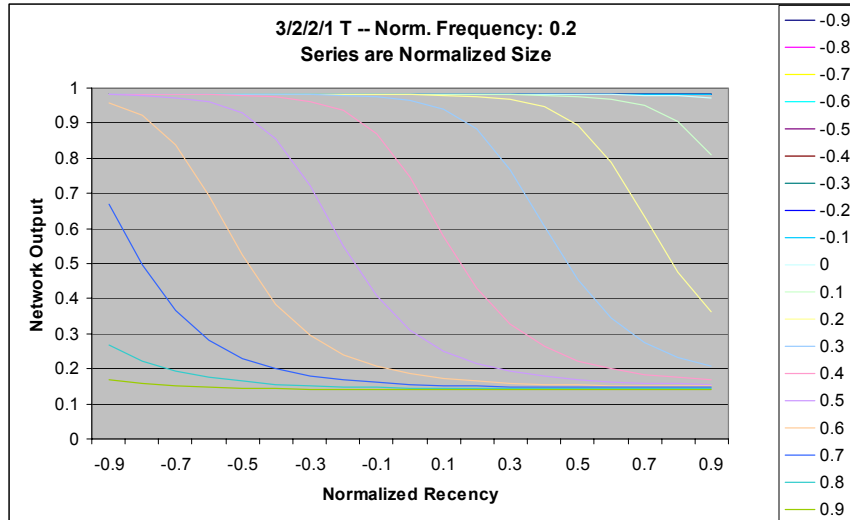


Figure 14. 3/2/2/1 T output for changing recency and size with a frequency of 0.2.

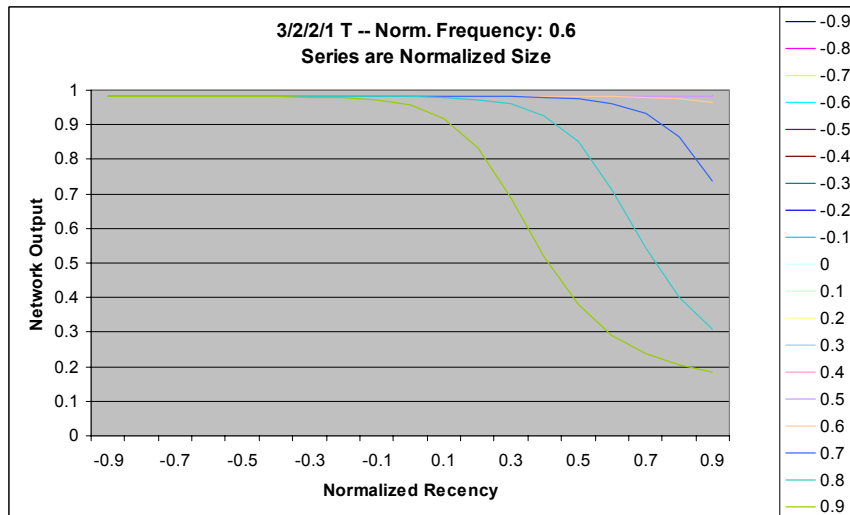


Figure 15. 3/2/2/1 T output for changing recency and size with a frequency of 0.6.

When the frequency reaches 0.9 the network output is flat-lined at 1.0 for all recency and size values, so the figure is not shown here. Frequency is the strongest factor in the 3/2/2/1 T network; the lowest possible frequency forces all output to below 0.5 except with the lowest possible values for both recency and size and the highest possible frequency forces all output above 0.5 without exception. Network output decreases for increases in either size or recency and the decrease in output follows a sigmoid shape for changes in recency with a fixed size and frequency when frequency is not suppressing the

shape. The 3/2/2/1 T network is more lenient with classification requirements for smaller objects. Small objects require less frequency and more recency to be considered un-cacheable than larger objects. However, the network shifts consideration from size to recency suddenly in a manner similar to the 2/2/2/1 network.

To summarize, the 2/2/2/1 network had the worst performance. It was characterized by balanced weighting of frequency and recency but with outputs tending towards extreme high or low values with a sudden classification shift between the two. The network's decision making is weak because the output is heavily clustered around 0.0 and 1.0, thus little to no distinction is made between items on one of the tails of the sigmoid. The 2/5/5/1 network had the best overall performance. It was characterized by considering frequency foremost with recency used as an aging factor. Additionally, the rate of change due to recency decreased with increasing frequency. The 2/5/5/1 network's performance may be attributed to its additional subtlety in mapping input to output. It does not have the same loss of distinction due to extended tails as the 2/2/2/1 network. Also, the change in output as recency changes assumes different shapes for different frequency values rather than simply shifting the points of inflection. Finally, the 3/2/2/1 network was generally unremarkable but excelled for hit rate with a high mark of 0.2 GB. It was characterized by the use of frequency as the overriding factor and a preference for classifying small objects as cacheable. The output was heavily clustered in a manner similar to the 2/2/2/1 network. The 3/2/2/1 network targets hit rate over byte-hit rate by virtue of its size consideration; choosing many small items attempts to maximize hit-rate whereas choosing fewer large items attempts to maximize byte-hit rate. The network likely suffers from clustering problems similar to the 2/2/2/1 network, but

may gain an advantage for lower high marks because it will not “plug” the cache with a very large object.

7. SUMMARY

Web proxy cache replacement has received considerable attention in the literature. Strategies used in local disk caching, such as LRU and LFU, comprise the base of most strategies proposed for proxy cache replacement. However, algorithms have been developed for web proxy caches specifically because the traffic patterns seen by a web proxy server vary from those seen in local caches on significant characteristics such as variation in object size and locality of references. The respective performances of the replacement algorithms are heavily dependant on metric and workload. These algorithms tend to either have assumptions about workload characteristics built in or include tune-able parameters to control which assumption(s) the algorithm favors.

Artificial neural networks are a model with a structure consisting of many nodes, often arranged in layers, and weighted connections between the nodes which seek to imitate the processing procedure of actual neural networks. MLP model neural networks are appropriate for web proxy caching because they are able to learn by example and generalize knowledge gained through training. The weights can be set to appropriate values through the process of supervised learning, wherein the network is trained against known input/output pairs and the weights are adjusted accordingly until the network converges to presenting correct output for all patterns in the training set. If the network is not over-trained, it should then be able to generalize reasonably to patterns outside the training set.

NNPCR is a novel web proxy cache replacement scheme which incorporates a neural network. The network is a two-hidden layer feed-forward artificial neural network which falls under the MLP model. NNPCR sets the connection weights through supervised learning with back-propagation. NNPCR trains the network using subsets of the trace data sets. Additionally, it uses separate subsets of the trace data sets for verification of network generalization. Error is calculated by using the known classification of each request for target output. Once the neural network has been created and properly trained, NNPCR is used to handle cache replacement decisions. All candidate replacement objects were rated by applying either frequency and recency information alone or those parameters plus size, across the inputs of the network. An object is selected for replacement based on the rating returned by the neural network. NNPCR aims to take advantage of neural networks' universal approximation and generalization properties. The neural networks created by NNPCR were able to classify both training and verification sets with CCR values in the range of .85 to .88, which indicates effective generalization. NNPCR derives workload assumptions from the training data (partial workloads) rather than holding a static set of assumptions or requiring the user to set parameters that dictate which assumptions behavior will reflect.

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. Furthermore, we have shown that such a network is able to effectively generalize to other workloads in a similar environment. We have presented simulation results which suggest this approach is a promising approach to web proxy cache replacement. Finally, we have explored the input-output mapping of several neural networks and identified

characteristics of these mappings that are potential causes for weak versus strong performance.

8. FUTURE WORK

NNPCR has shown that two hidden layer MLP neural networks can be easily and successfully trained for web proxy cache replacement using the basic back-propagation algorithm with momentum. However, the neural networks were frequently outperformed by the basic LFU algorithm. The results are still encouraging because back-propagation is a relatively simplistic, albeit the most commonly used, method. The literature contains numerous variations on back-propagation, constructive methods for determining network structure, pruning methods for eliminating redundant or conflicting nodes, and applications of more general strategies, such as genetic programming and classical optimization techniques, to neural networks. Furthermore, many of these techniques are not mutually exclusive. Future work should investigate the use of more advanced training techniques in an attempt to enhance performance over the basic back-propagation algorithm.

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 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.

REFERENCES

- [1] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374-398, 2003.
- [2] C. Stergiou and D. Siganos, "Neural Networks," [Online document] [cited Sept. 9, 2005] Available WWW: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.
- [3] L. Fu, "Knowledge discovery based on neural networks," *Communications of the ACM Archive*, vol. 42, no. 11, pp. 47-50, 1999.
- [4] P. P. van der Smagt, "A comparative study of neural network algorithms applied to optical character recognition," in *Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (ACM Press, 1990, vol. 2, pp. 1037-1044)
- [5] B. Dasgupta, H.T. Siegelmann, and E. Sontag, "On a learnability question associated to neural networks with continuous activations," *Annual Workshop on Computational Learning Theory*, pp. 47-56, 1994.
- [6] R. D. Reed and R. J. Marks II, *Neural smithing: supervised learning in feedforward artificial neural networks*. The MIT Press, 1999.
- [7] R. P. Lippmann, "An introduction to computing with neural nets," *ASSP Magazine*, pp. 4-22, April 1987.

- [8] A. Lapedes and R. Farber, "How neural nets work," in *Neural Information Processing Systems*. (American Institute of Physics, 1988, pp. 442-456)
- [9] C. C. Aggarwal, J. L. Wolf and P. S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowl. Data Eng.* 11, pp. 94-107, Jan. 1999.
- [10] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating content management techniques for Web proxy caches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 4, pp. 3-11, 2000.
- [11] H. Bahn, K. Koh, S. L. Min and S. H. Noh, "Efficient replacement of nonuniform objects in Web caches," *IEEE Comput.* 35, pp. 65 – 73, June 2002
- [12] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," in *Proceedings of USENIX Symposium on Internet Technologies and Systems*. (1997, pp. 193-206)
- [13] H. Khalid, "A new cache replacement scheme based on backpropagation neural networks," *ACM SIGARCH Computer Architecture News*, vol. 25, no. 1, pp. 27-33, 1997.
- [14] J. Pomerene, T.R. Puzak, R. Rechtschaffen and F. Sporacio, "Prefetching Mechanism For a High-Speed Buffer Store," US Patent, 1984.
- [15] H. Khalid, "Performance of the KORA-2 cache replacement scheme," *ACM SIGARCH Computer Architecture News*, vol. 25, no. 4, pp. 17 – 21, 1997.
- [16] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Transactions on Neural Networks*, 4(1), pp. 136-141, 1993.

- [17] S. C. Rhea, K. Liang and E. Brewer, “Value-based Web caching,” in *Proceedings of the 12th International Conference on World Wide Web*. (ACM Press, 2003, pp. 619-628)
- [18] R. Caceres, F. Douglis, A. Feldmann, G. Glass and M. Rabinovich, “Web proxy caching: the devil is in the details,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 11 – 15, 1998.
- [19] L. Rizzo and L. Vicisano, “Replacement policies for a proxy cache,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 158 – 170, 2000.
- [20] G. Ganger, B. Worthington, Y. Patt and J. Bucy, “The DiskSim Simulation Environment,” [Online document] Sept. 2005 [cited Sept. 16, 2005] Available WWW: <http://www.pml.cdu.edu/DiskSim/>
- [21] “IRCache Home,” [Online document] [cited Sept. 5, 2005] Available WWW: <http://ircache.net/>