Branch Prediction using Artificial Neural Networks

Rohan Kumar
Arvind Nithrakashyap
Jitendra Padhye
Jayavel Shanmugasundaram

Abstract

Accurate branch prediction has become very critical for improving the performance of the present day superscalar and superpipelined architectures. Mispredicted branches imply a wastage of tens of cycles and have proved to be a bane of the present day machines. In this project, we have analyzed the strength of artificial neural networks as a technique for branch prediction. We have experimented with three neural network architectures and compared their prediction performance with that of static prediction techniques. The results of our study show that artificial neural networks, especially feed-forward neural networks, perform significantly better than static branch prediction techniques and that they can be used as a low-overhead branch prediction mechanism in modern day processors.

Contents

1 Introduction

Pipeline hazards are a major factor of performance loss of a processor [12]. They reduce the speedup gained by pipelining by preventing the execution of next instruction in the stream, thereby stalling the pipeline. There are basically three kinds of pipeline hazards:

- *Structural Hazards* which arise due to hardware resource conflicts.
- *Data Hazards* which arise when an instruction depends upon the data produced by a previous instruction which has not completed execution.
- *Control Hazards* which arise due to branches in the code which change the Program Counter.

Various techniques have been proposed in the literature for dealing with each of the above hazards. Among the three, the Control Hazards, have proven to be a bane for the present day superscalar and superpipelined processors. Thus, an increase in the branch prediction accuracy would lead to a significant increase in performance because pipelines would not have to be flushed many times due to control hazards.

There are a number of mechanisms which have been proposed to ameliorate the effect of control flow changes. They can be broadly classified into static [6, 9] and dynamic schemes [9, 16]. Static schemes only consider some fixed attributes at compile time and make only a single prediction for every branch for the entire execution of the program. On the other hand, dynamic schemes consider the recent execution history of the branch at run time and make a decision. Static schemes are easy to implement in the hardware or the compiler. Dynamic schemes require more hardware and compiler support. Another method that has been proposed in the literature to reduce the branch penalty and is called the Delayed branch approach. The sequential successors are in the branch delay slots (i.e., the slots in the pipeline following the instruction). These instructions are executed whether or not the branch is taken. The job of the compiler here is to select the successor instructions rather than predicting the outcome of the branch.

While static prediction mechanisms, which are mostly profile based methods, accurately predict 80 - 90% of the branches, the performance of modern computer architectures would be substantially increased by higher branch prediction accuracies. In this context, dynamic branch prediction mechanisms seem to be a solution to this problem and this is an active research area. In this project, we have analyzed the usefulness of various artificial neural network architectures [8, 13] as a dynamic branch prediction mechanism. The architectures we have considered are (a) Feed Forward Neural Networks using logistic units, (b) Recurrent Neural Networks and (c) Feed Forward Neural Networks using Associative Reward Penalty (ARP) [1] units. We have implemented the above architectures and have tested their branch prediction performance by experimenting with the standard SPEC benchmarks.

The remainder of the report is organized as follows. Section 2 describes the motivation behind using neural networks for branch prediction and also the various neural net architectures that

we have experimented with. Section 3 describes our experimental setup and the benchmarks used in our experiments. The results of our experiments are presented and analyzed in section 4. Section 5 concludes the report and presents areas for future work.

2 Neural Networks for Branch Prediction

2.1 Why Neural Networks ?

Neural Networks [8, 13] have the capacity to generalize and learn patterns using a limited amount of training data. They can utilize the patterns gleaned from the training data to predict values on unknown data sets having similar characteristics.

A neural network is composed of a number of nodes or units connected by links. Each link has a numeric weight associated with it. Weights are primary means of long term storage in neural networks and learning usually takes place by updating the weights. Some of the units are connected to the external environments and can be designated as input or output units. To build a neural network, one must first decide, the number of units that are to be used, the units that are appropriate and the way the units are connected together to form the network. The weights of the network are then initialized and trained using a learning algorithm applied to a set of training examples for the task.

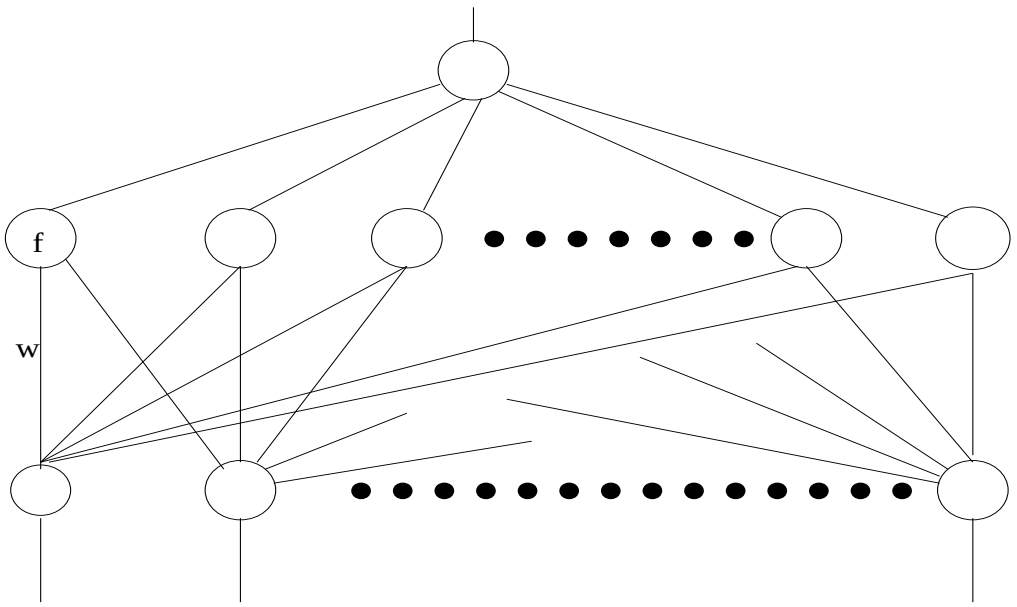
There are a number of different neural net architectures mentioned in literature [7] with different computational properties and are broadly classified into feed forward and recurrent networks. The links between units in a feed forward network form an acyclic graph while the links in recurrent networks can form a cyclic graph.

Neural networks have been employed in temporal sequence processing [5, 10] tasks before, with reasonably successful results [14]. Since branch prediction can also be viewed as a temporal sequence processing task, in which past branch results are used to predict future branch results, we decided to investigate the power of neural networks for branch prediction tasks.

2.2 Neural Network Architectures

Based on papers on temporal sequence processing [5, 10] in the literature, we selected the following neural network architectures for branch prediction.

- **Feed Forward Neural Networks:** These are the simplest form of neural networks in which the output of the neural network depends only on the current input.
- **Recurrent Neural Networks:** These neural networks can have feedback (in the form of cycles feeding the output back as input). This enables recurrent neural networks to maintain state and so the output is not only a function of the current input but also of past input.



w = weights of the links

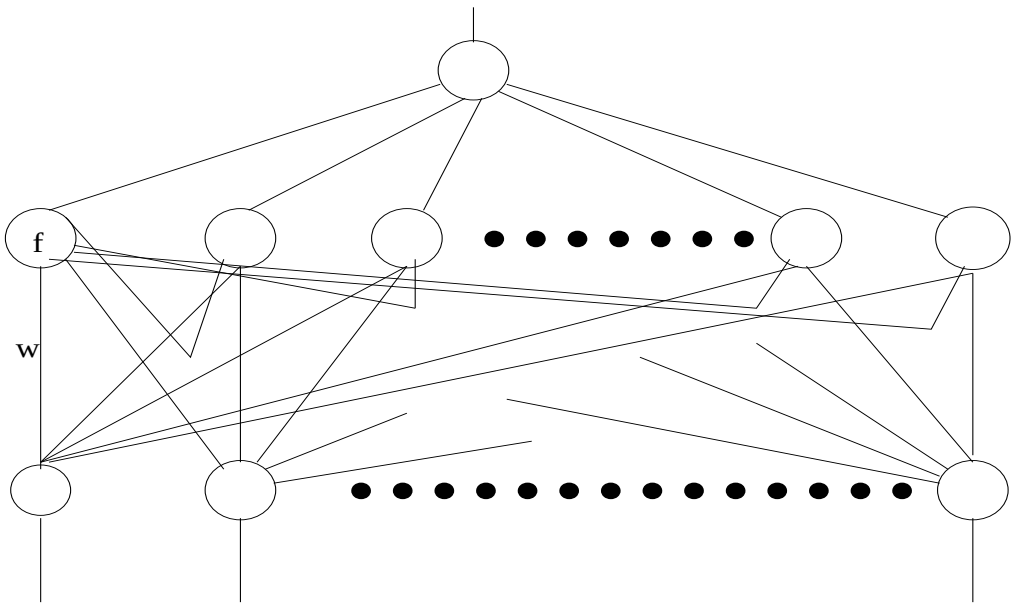
f = function implemented at the nodes

Figure 1: Feed Forward Architecture

- **Feed Forward Network with ARP Units:** These are like the traditional feed forward networks. However, the units can also be probabilistic (ARP) unlike the deterministic units in traditional feed forward networks.

2.2.1 Feed Forward Architecture

The Figure 1 describes the feed forward neural net architecture. As shown in the figure, it has some input nodes, some hidden nodes and output nodes which are connected to each other with links to form a directed acyclic graph. The connecting links have weights. The input to the neural network is fed through these input nodes. As described later, one possible input to the nodes is the past history of branch output values. Each input node is connected to the hidden nodes. The value at each node is multiplied by the weight on each out going link and is passed on to the node on the other side of the link. All the internal nodes have a logistic function implemented which operates on all the input values and produces an output which is passed on. The function that is implemented at each of these node is $\frac{1}{1+e^x}$. Finally, when the output node receives all its weighted input values, it computes the output using the logistic function. The output of this node is the final predicted output of the neural net. The difference between the predicted and the actual output is back propagated [8] to input nodes so that they adjust the link weights (depending on the variation in the predicted value and the actual value) for a better prediction. The feed forward neural network was implemented in C++.



w = weights of the links

f = function implemented at the nodes

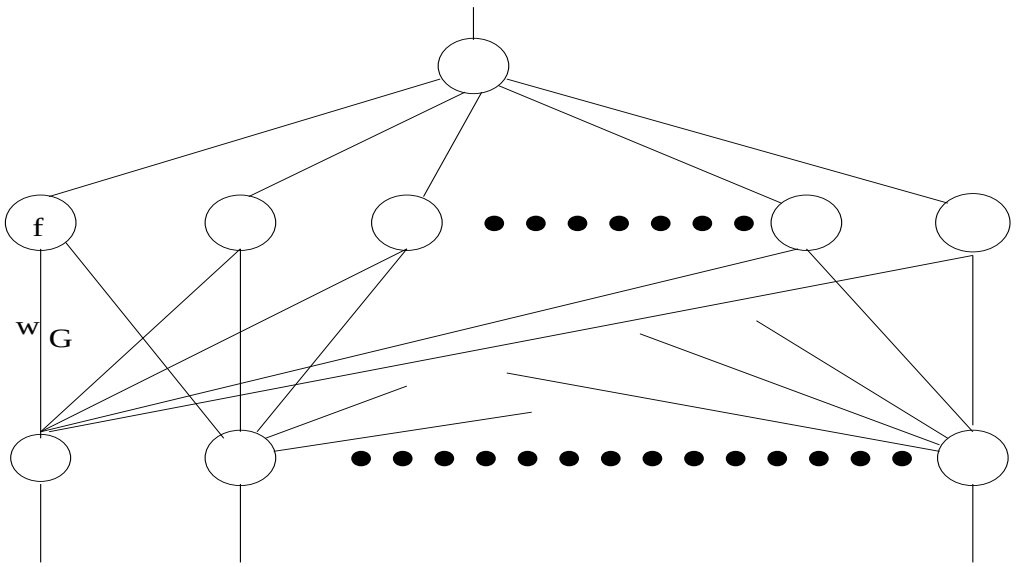
Figure 2: Recurrent Architecture

2.2.2 Recurrent Architecture

The Recurrent architecture is shown in the Figure 2. As shown in the figure, the main difference between feed forward and recurrent architectures is that the recurrent architectures allow for cycles within the internal nodes. This feedback in the form of cycles helps the neural net maintain state information (consisting of past inputs and outputs) which could aid in more accurate predictions. The obvious overhead is the complication involved in implementing the cycles. Other concepts like input nodes, functions etc., are similar to the feed forward neural networks. Our implementation of recurrent neural networks uses the Real Time Recurrent Learning (RTRL) algorithm [15] for updating weight of links.

2.2.3 Feed Forward Network using ARP Units

The Figure 3 describes the Feed Forward Network using ARP Units. These networks are a variation of the traditional feed forward networks in which the logistic units at the hidden node layer are replaced by ARP units [1] and eligibility traces [2] are added to edges feeding the ARP units. The ARP units are stochastic units whose output is either 0 or 1 with a probability computed by the logistic function. This probabilistic operation reduces the chances of the network getting stuck at local minima during training. The second feature which differentiates these networks from traditional neural networks is the notion of eligibility traces. Eligibility traces keep a compact representation of the past behavior of the network and use it to update the weights of edges. The eligibility traces are exponentially decaying



w = weights of the links

f = function implemented at the nodes

G = Eligibility traces function on the links

Figure 3: Feed Forward with ARP Units

and so some small component of past events always remain. This gives the network an "infinite horizon" to look at during training. Our implementation of these networks was in C++.

3 Experimental Setup

The experimentation process involved the following steps

- Selecting benchmarks.
- Gathering data on the selected benchmarks through ATOM.
- Implementing the neural network (three architectures).
- Training and testing the neural network on the data.
- Assimilating results.

3.1 Benchmarks Considered

Three benchmarks were chosen for these experiments:

- **Compress95**: It compresses and decompresses a 1 MB file 20 times using the Unix compress utility (SpecInt-95 benchmark)
- **Perl**: does string processing, generating anagrams and prime numbers, using the PERL language (SpecInt-95 benchmark).
- **Tomcatv**: is a 2D mesh benchmark that generates a 2D boundary-fitted coordinate system around a geometric region (SpecFP-95 benchmark).

We selected the above benchmarks, as they seemed to represent a diverse range of applications.

3.2 ATOM

We used ATOM [4] for our simulations. ATOM is a tool developed at Digital Equipment Corporation, that allows the user to instrument object code, without modifying the object code. This process can be used to gather data about the execution of benchmarks. The instrumentation routines need to be supplied by the user. The data needed for these experiments were the branch traces (i.e whether the branch was taken was not, each time the branch was accessed by the object code). This data was then passed on to the neural nets as training and testing data sets.

3.3 Architectural Parameters

Some of the parameters that we varied for these experiments were:

- **Branch History:** The number of past branches that the neural net used to predict the outcome of the next branch.
- **Branch Grouping:** The number of branches that are grouped together in the input to the neural network.

3.3.1 Branch History

The extent of the history that a neural network uses to make predictions can be varied. In this case, this is represented by the number of previous branch results that are used. For our experiments we used history lengths of 1, 2, 4 and 8. For example, if a history of length 2 is used, the neural network uses the previous two branch results to make the prediction. Hence history is a measure of how far back the network looks at to make the prediction. Our hypothesis was that increase in history would lead to better results, since the network would have more information for its predictions.

3.3.2 Branch Grouping

Branch grouping represents the number of branches that are grouped together as input to the neural network. For our experiments, we used groups of size 1, 2 and 4. As the group size increases, the network has less information per branch for a given history length. For a single branch the history would represent the results for that branch alone. However with increasing group sizes, the history would be divided among the results for the various branches in the group. Hence, our hypothesis was that increase in group size would lead to poorer prediction, for a given size of the history. However, grouping branches has the advantage of providing a cheaper implementation scheme.

3.4 Neural Network Parameters

One of the important steps in the use of neural networks, is the determination of the learning rate, momentum and the random seed. We experimented with different values for these parameters and determined the values for which the neural network performed well. These values were then used for the experiments described in the previous section.

3.5 Experimental Methodology

The experiments that we performed were as follows:

- For each benchmark, two branches were chosen as individual branch inputs to the neural network (i.e no grouping of branches). We attempted to select branches that did not have regular patterns, since it has been shown that neural networks are good at recognizing regular patterns. An observation that we made was that among the data sets we had collected, most branches exhibited regular patterns. Neural networks be very effective in predicting such branches.
- We also conducted experiments that studied the effect of grouping on branch prediction. We considered groups of 1, 2 and 4. The groups were picked so that the larger group contained all the branches in the smaller groups. Thus the branch in group-1 was in group-2 and both the branches in group-2 were in group-4.
- While training the neural networks, we had to watch out for over-fitting. When the neural networks are trained for too long with the training data set, they tend to recognize patterns that are specific only to the training set. Thus, their accuracy with the test set tends to decrease. In other words, the neural network tends to "over-fit" the training set thus losing their effectiveness in modeling the general case. Figure 4 illustrates this situation where the percentage of correct predictions starts decreasing after about 130 epochs. Hence, the training should be stopped at this point.

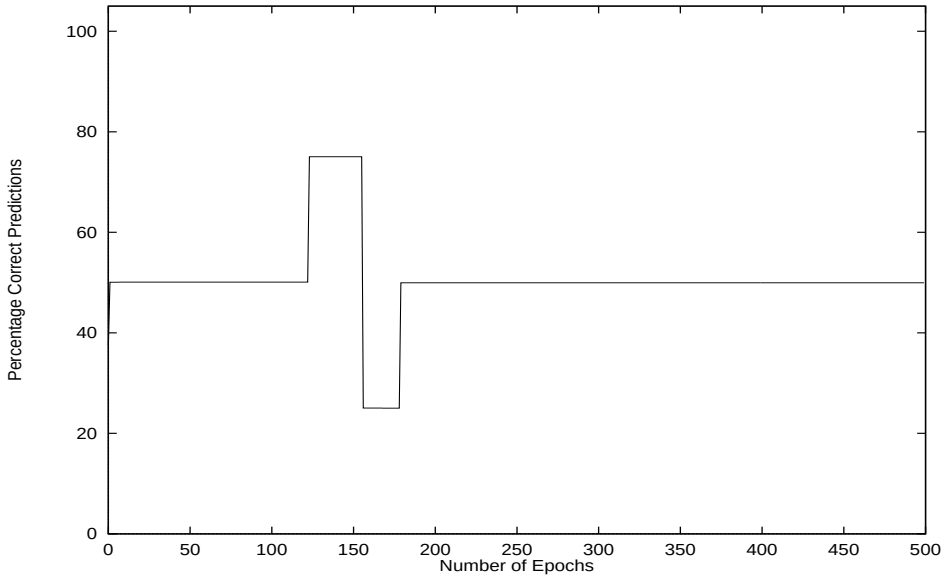


Figure 4: Over Fitting Curve

4 Branch Prediction Results

In this section, we present and analyze the results of our experiments. In section 4.1, we compare the branch prediction performance of the three neural network architectures, using the static prediction as the baseline. In sections 4.2 and 4.3 we describe the effect of history and grouping, respectively, on branch prediction. Finally in section 4.4, we outline why we believe that feed forward neural networks are the most feasible neural network architecture for branch predictions.

4.1 Comparison of the Architectures

Figures 5 and 6 show the prediction results of feed forward and recurrent neural networks respectively on a branch in the Compress95 benchmark. From this figure, it is easy to see that the branch prediction using feed forward and recurrent neural networks performs significantly better than static prediction. The graphs presented here are representative for the compress95 benchmark in the sense that for almost all the branches in the compress95 benchmark for which we trained the neural networks, we got similar results. However, we observed that the feed forward neural network architecture employing ARP units performed exactly like static prediction (i.e., it output only one value, depending on the frequency of the branch being taken or not taken). Thus, only the feed forward and recurrent neural architectures were actually able to pick up the trends in the data while the feed forward neural network with ARP units did not learn much. Further, the prediction capabilities of feed forward and recurrent neural networks are comparable, even though the recurrent neural network has state.

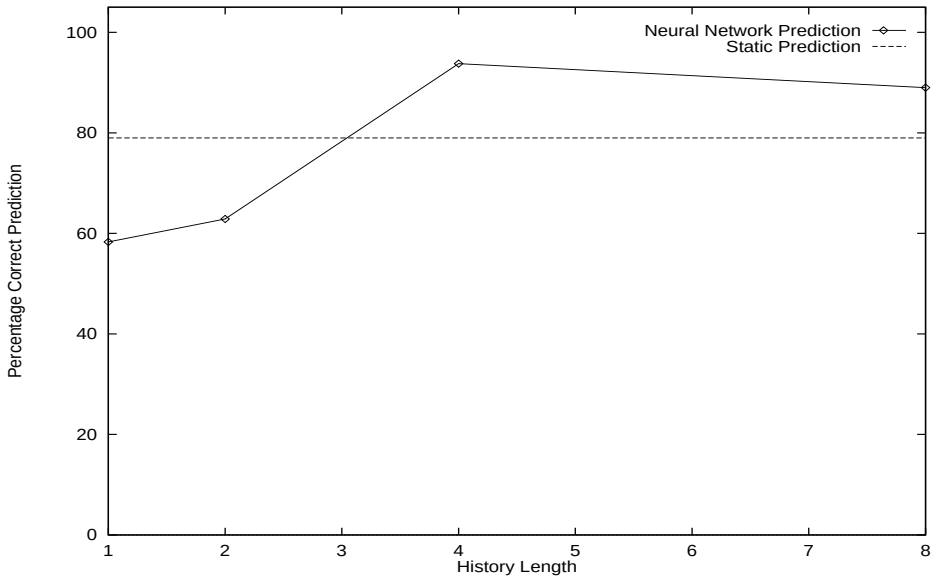


Figure 5: Branch 1 (Compress95): Prediction using Feed Forward NN

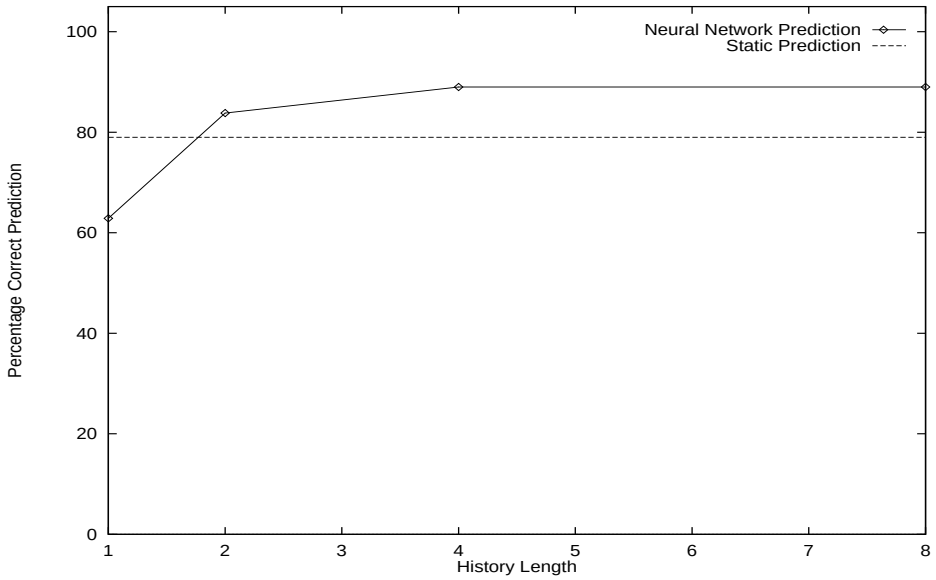


Figure 6: Branch 1 (Compress95): Prediction using Recurrent NN

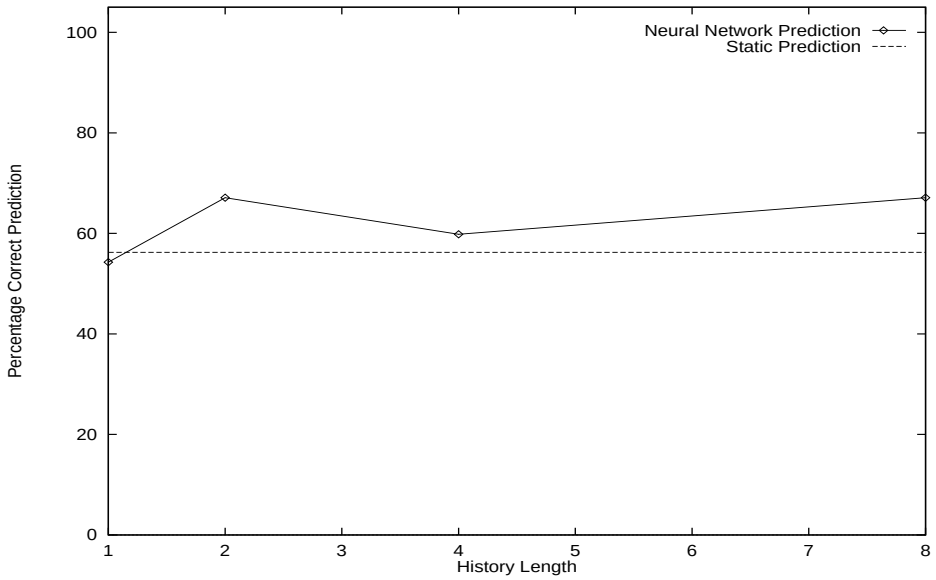


Figure 7: Branch 1 (Perl): Prediction using Feed Forward NN

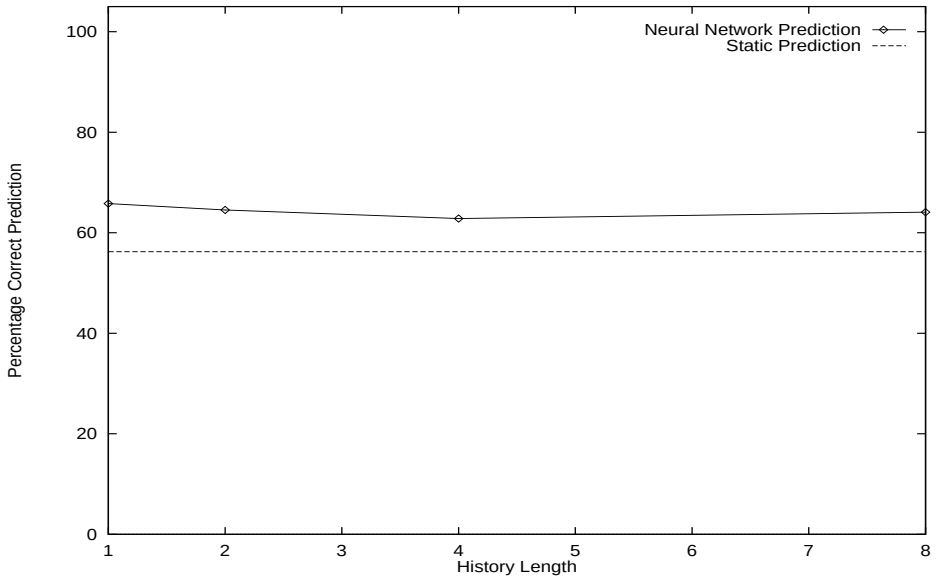


Figure 8: Branch 1 (Perl): Prediction using Recurrent NN

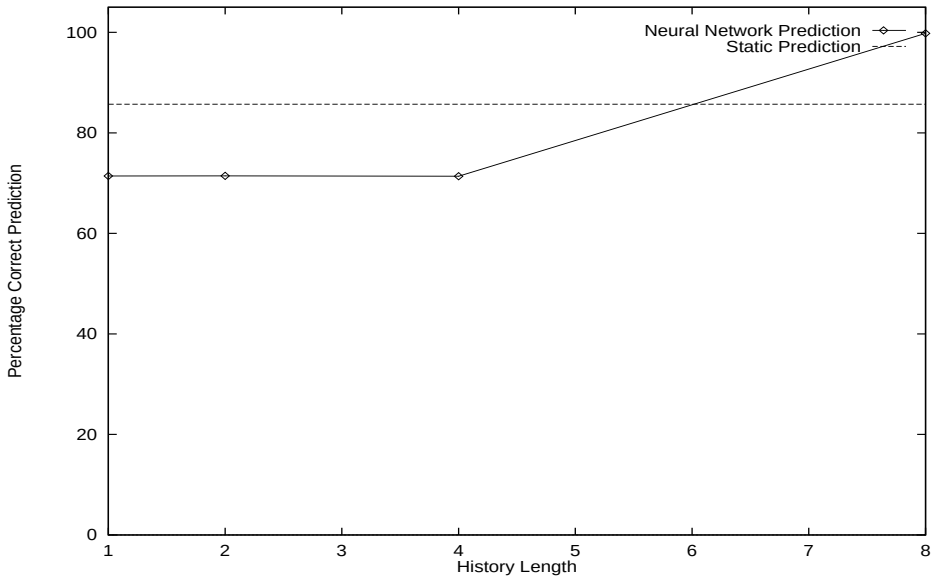


Figure 9: Branch 1 (Perl): Prediction using Feed Forward NN

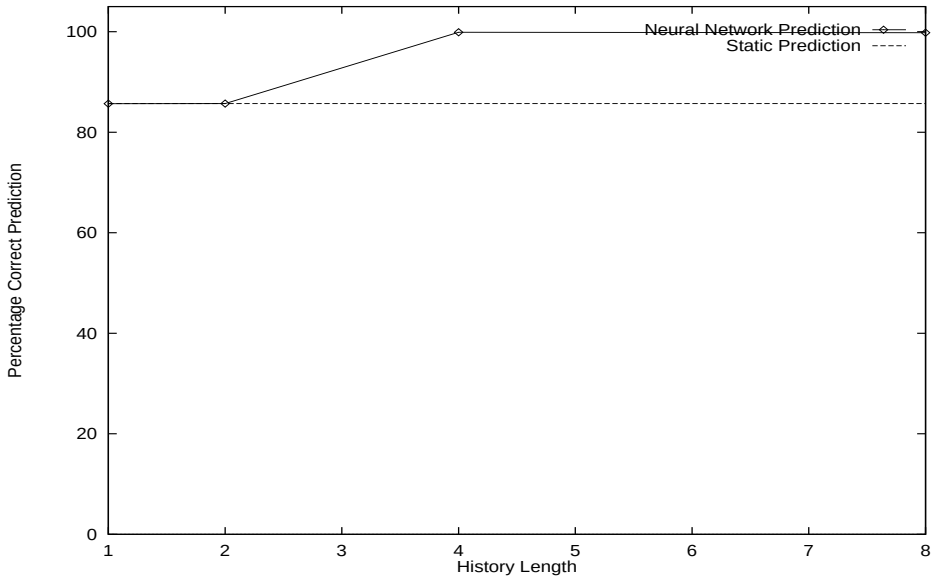


Figure 10: Branch 1 (Tomcatv): Prediction using Recurrent NN

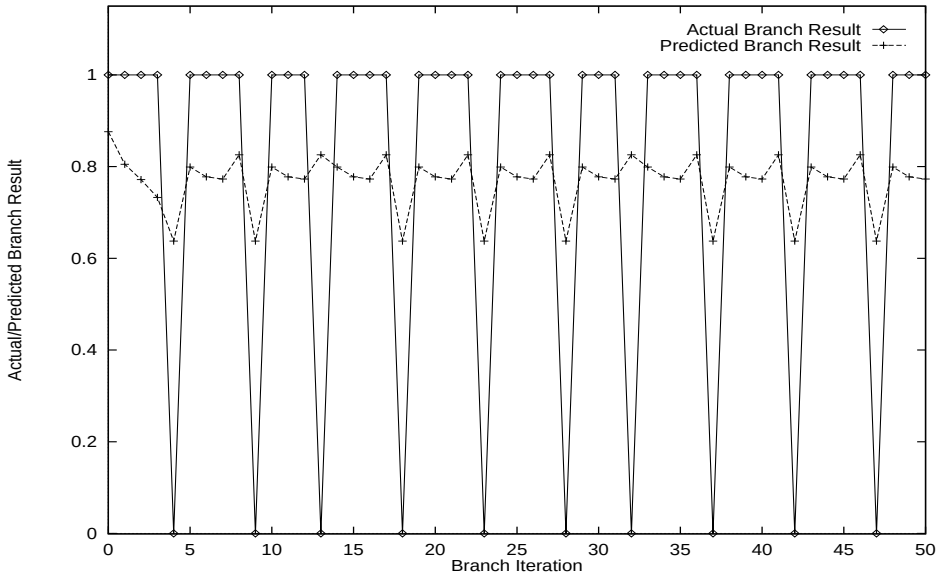


Figure 11: Actual Branch Results vs Neural Network Prediction

We obtained similar results for the Perl and Tomcatv benchmarks, as shown in figures 7, 8, 9 and 10. Again, only the feed forward and recurrent neural network architectures performed significantly better than static prediction. Also, we noticed that most branches in benchmarks were very regular (for example, most were periodic with a period of at most 6, like those in figure 5 and 9) and neural networks performed very well in predicting these branches. Only very few branches were more random (like the one in figure 7) and in these cases, even though the prediction performance was not very good, it was comparable to that of static prediction.

4.2 Effect of History

The length of the history played a crucial role in the prediction performance of neural networks. For example, in figure 11, we can see how the neural network learns and adjusts to changes in branch patterns using past history values. With sudden changes in branch values, the neural network mispredicts once, but very quickly realizes the change and adjusts itself to it thus predicting accurately. In this case, the neural network makes use of the history (which is of length 8 and greater than the periodicity of the branch) and of the pattern it has learnt in order to make predictions.

We also observed that with an increase in the length of the history, the prediction accuracy also increases (see figures 5 through 16). This is in accordance with the intuition because the higher the knowledge of the neural network regarding a branch, the better should be the prediction accuracy. It can also be seen that the accuracy of prediction with neural networks is much higher than static prediction for higher history lengths. The only exception seems to be Figure 14 where the static prediction accuracy is always better than the prediction of the neural net. This is because this particular branch that we selected was extremely random

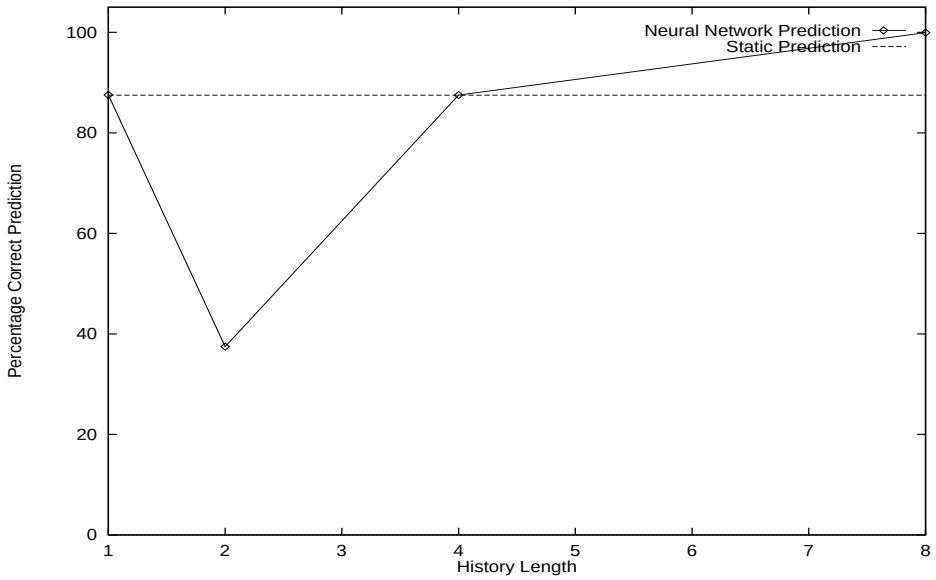


Figure 12: Branch 2 (Compress95): Prediction using Feed Forward NN

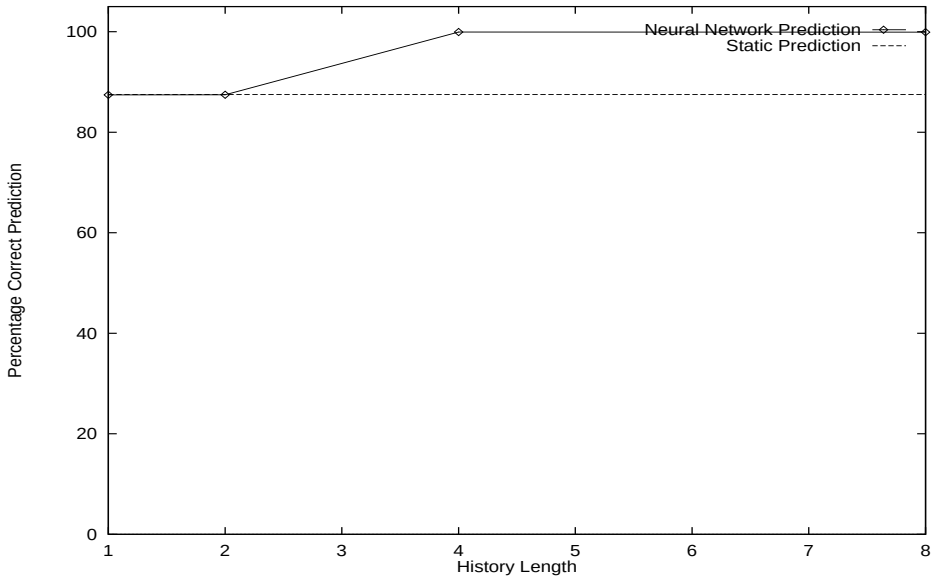


Figure 13: Branch 2 (Compress95): Prediction using Recurrent NN

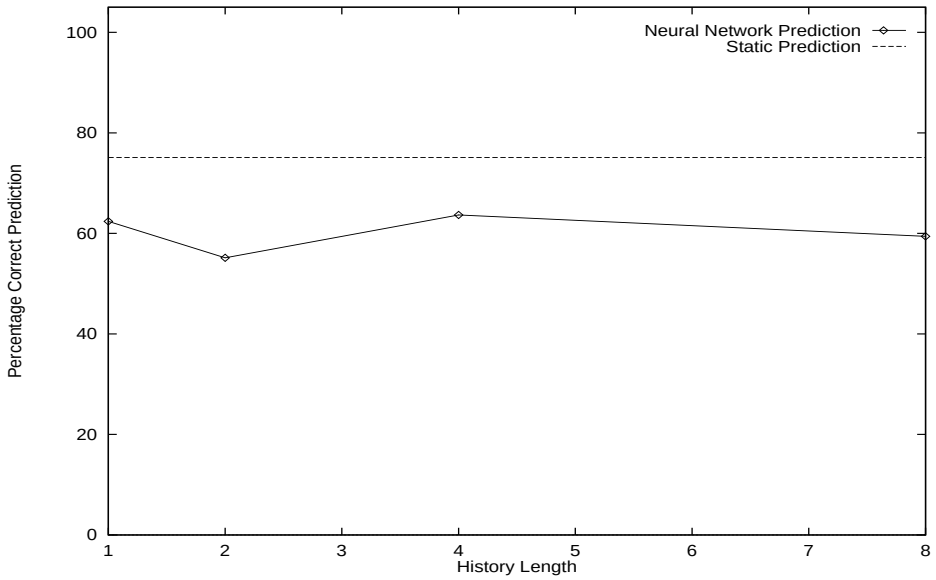


Figure 14: Branch2 (Perl): Prediction using Feed Forward NN

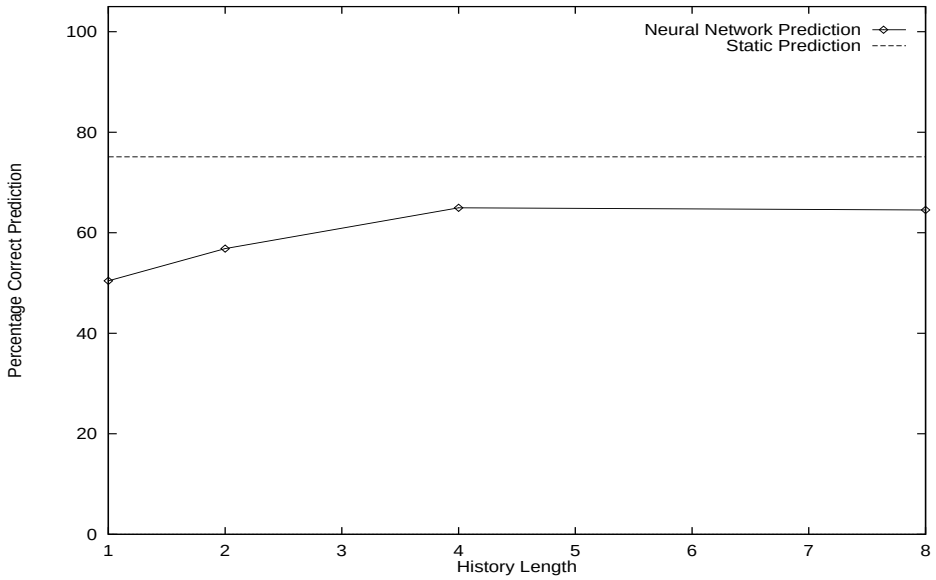


Figure 15: Branch 2 (Perl): Prediction using Recurrent NN

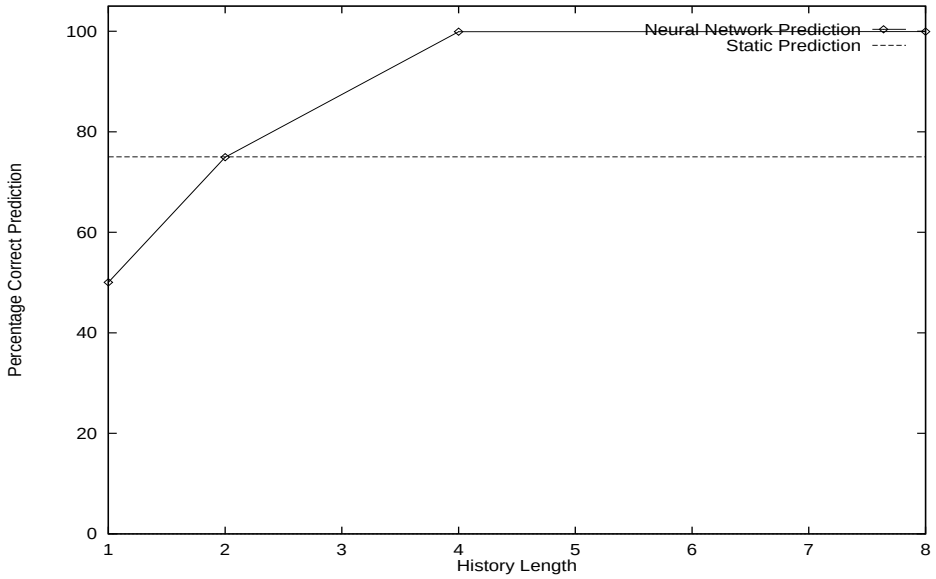


Figure 16: Branch2 (Tomcatv): Prediction using Feed Forward NN

in its behavior and hence the neural network could never catch up with the data.

4.3 Effect of Grouping

Figures 17 to 22 study the effect of grouping branches on neural network prediction performance. Figure 17 shows the prediction performance of feed forward neural networks for Branch 3 of Compress95. Figure 18 show the prediction performance of feed forward neural networks when Branch 3 is combined with three other neighboring branches. We notice that the prediction performance steadily drops with an increase in the size of the group. This is to be expected in this case because the branches in the group did not have too much correlation with each other. Thus, because one neural network is being used to predict more than one branch, the prediction accuracy falls. Similar results can be observed in figures 19 and 20.

In figure 22, we notice that prediction performance does not monotonically decrease with an increase in the size of the group. This is because, in the group of size two, the two branches did not have much correlation, while in the group of size four, three of the four branches were highly correlated. Thus, correlated branches increase accuracy, which is in accordance with the study in [11].

4.4 Preferred Architecture for Branch Prediction

The results show that the performance of Feed Forward and Recurrent neural networks (in terms of branch prediction accuracy) are comparable. However, Recurrent networks take large amounts of time to train as compared to Feed Forward networks. Recurrent neural

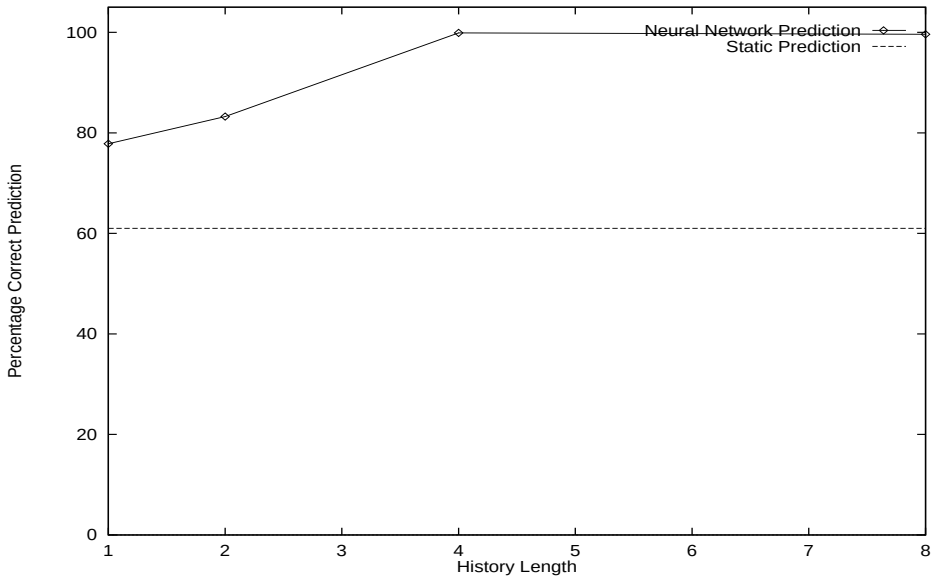


Figure 17: Branch 3 (Compress95): Prediction using Feed Forward NN

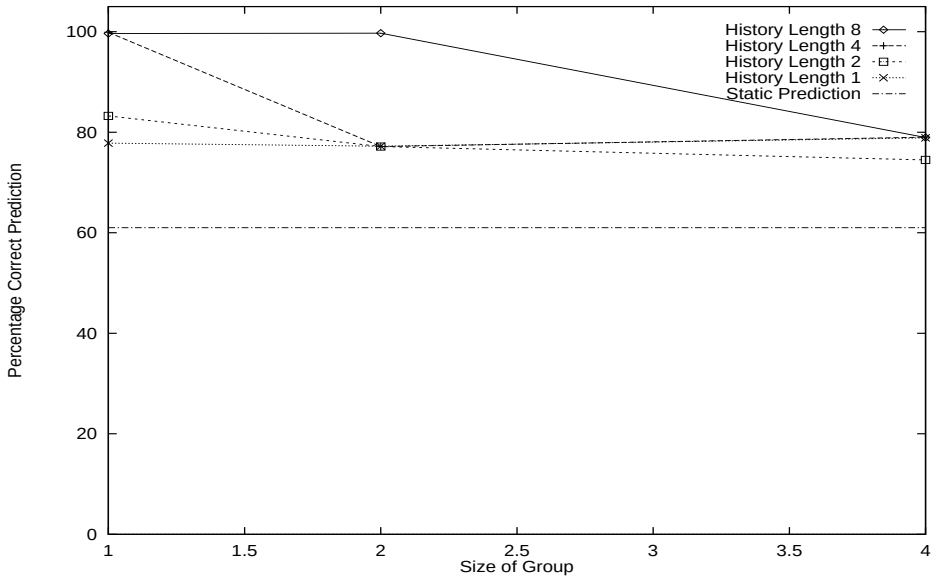


Figure 18: Grouping Branch 3 (Compress95): Prediction using Feed Forward NN

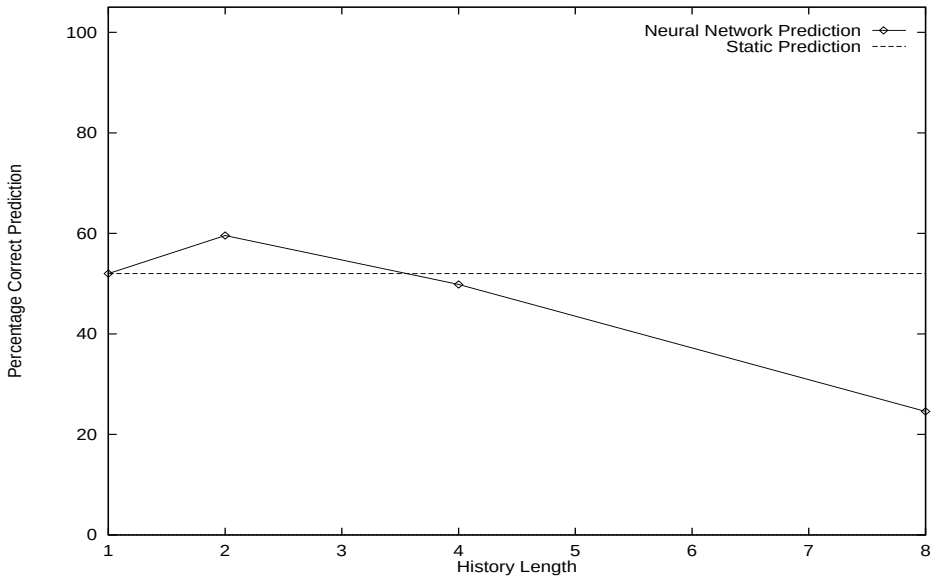


Figure 19: Branch 3 (Perl): Prediction using Feed Forward NN

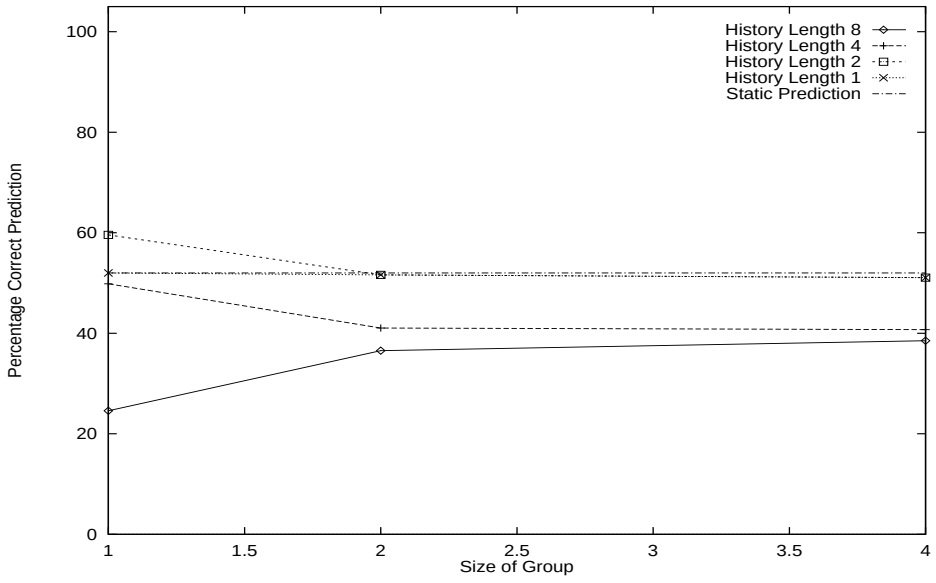


Figure 20: Grouping Branch 3 (Perl): Prediction using Feed Forward NN

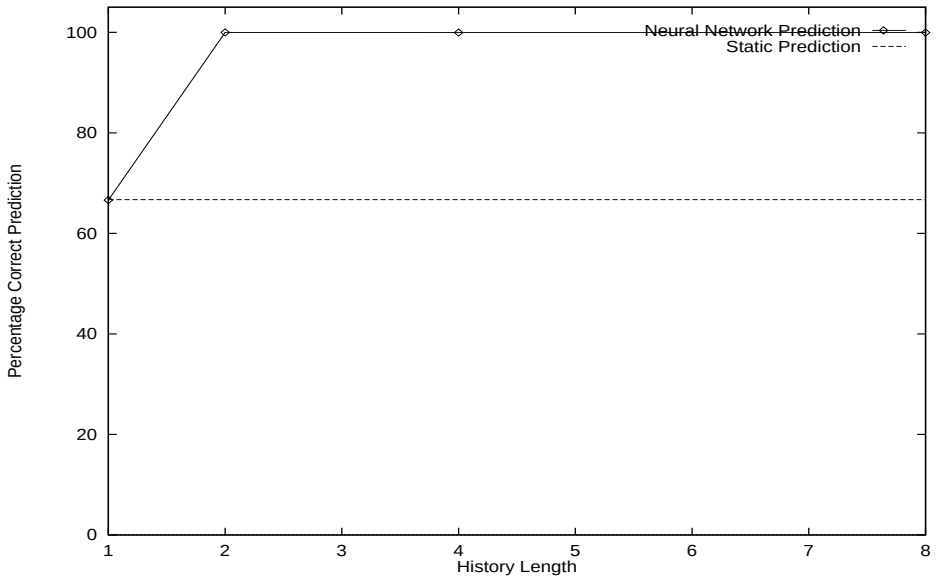


Figure 21: Branch 3 (Tomcatv): Prediction using Feed Forward NN

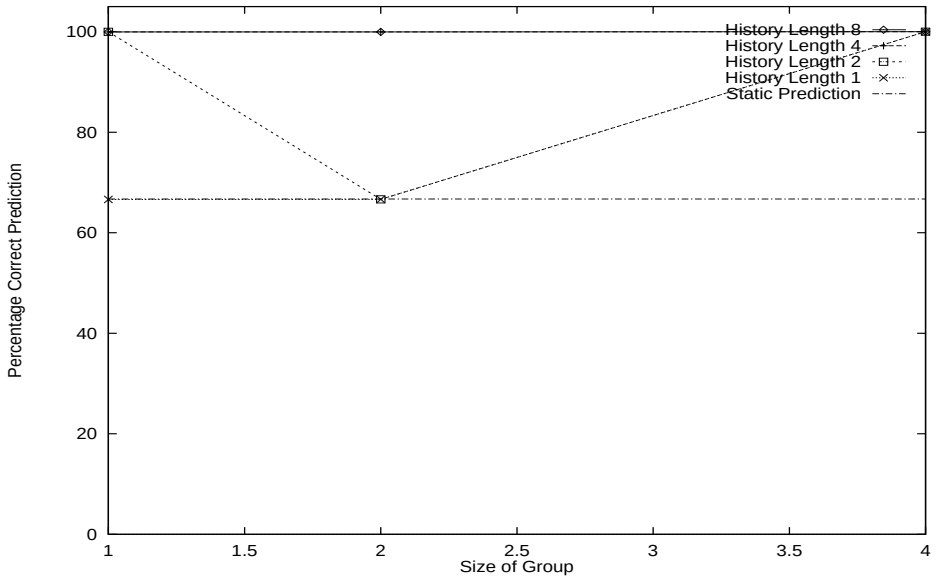


Figure 22: Grouping Branch 3 (Tomcatv): Prediction using Feed Forward NN

networks update their state during the prediction phase which may be a performance penalty relative to Feed Forward networks which just pass the inputs through the network and generate the output. Owing to these factors, Feed Forward neural networks may be more feasible, although they may require longer histories to predict as accurately as Recurrent neural networks. As mentioned earlier, feed forward networks with ARP units do not predict better than static prediction techniques and are hence not feasible.

5 Conclusions and Future Work

The experiments that were performed show that neural networks are very effective for branches that display regular patterns. They also perform well for branches that show certain irregularities (for example, varying step sizes) and display the ability to capture trends. If the branch trace happens to be random, neural networks do not prove to be very effective. Among the various architectures that were considered, Feed Forward and Recurrent neural networks produced comparable results, which are better than static prediction. However, as mentioned earlier, Recurrent networks require a long time to train. The performance of ARP units, on the other hand, is comparable to static prediction techniques. Hence, Feed Forward neural networks appear to be the most feasible architecture for branch prediction.

Only three benchmarks were used in this project, owing to time constraints. An extensive study with more benchmarks would be needed to make a clear statement about effectiveness of neural networks in branch prediction. Investigation is needed to determine ways in which neural networks can be efficiently integrated into a dynamic branch prediction framework. It would also be interesting to study the cost (in terms of time) of neural networks in such a framework.

6 Acknowledgment

We would like to thank Maram V. Nagendra Prasad for helping us with the choice of appropriate neural network architectures for branch prediction.

References

[1] Barto, A.G., Jordan, M.L., "Gradient Following Without Back-Propagation in Layered Networks," Proceedings of the IEEE First Annual International Conference on Neural Networks, June 1987.

[2] Barto, A.G., Sutton, R.S., Anderson, C.W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No. 5, Oct. 1983.

[3] Calder, B., Grunwald, D., "Branch Prediction Architectures for 64-bit Address Space," University of Colorado at Boulder Technical Report CU-CS-690-93, November 1993.

[4] "Program Analysis using ATOM Tools," Programmers Guide, Digital Equipment Corporation, Maynard, Massachusetts, March 1996.

[5] Elman, J.L., "Finding Structure in Time," Cognitive Science, vol. 14, pp. 179-211, 1990.

[6] Fisher, J., Freudenberger, S., "Predicting Conditional Branch Directions from Previous Runs of a Program," Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM, Boston, pp. 85-95, October 1992.

[7] Hertz, J.A., Krogh, A.S., Palmer, R.G., "Introduction to the Theory of Neural Computation," Santa Fe Institute Studies in the Sciences of Complexity, Lect. Notes vol. I, Addison-Wesley, 1991.

[8] McClelland, J.L., Rumelhart, D.E., "Explorations in Parallel and Distributed Processing," MIT Press, Cambridge, 1988.

[9] McFarling, S., Hennessy, J., "Reducing the cost of branches," Proc. 13th Symposium on Computer Architecture, Tokyo, pp. 396-403, June 1986.

[10] Mozer, M.C., "Neural Net Architectures for Temporal Sequence Processing," Predicting the future and understanding the past (Eds. A. Weigend and N. Gershenfeld), Addison-Wesley, 1993.

[11] Pan, S., So, K., Rahmeh, J., "Improving the Accuracy of Dynamic Branch Prediction using Branch Correlation," Proc. 5th International Conference of Architectural Support for Programming Languages and Operating Systems, October 1992.

[12] Patterson, D.A., Hennessy, J.L., "Computer Architecture: A Quantitative Approach," Second Edition, Morgan Kaufmann Publishers, San Fransisco, California, 1996.

[13] Rumelhart, D.E., McClelland, J.L., "Parallel Distributed Processing : Explorations in the Microstructure of Cognition," MIT Press, Cambridge, 1986.

[14] Shanmugasundaram, J., Nagendra Prasad, M.V., Gupta, A., "Temporal Data Mining," PROFIT Research Initiative Working Paper 97-31, Massachusetts Institute of Technology, 1997.

[15] Williams, R.J., Zipser, D., "Experimental Analysis of the Real-time Recurrent Learning Algorithm," Connection Science, vol. 1, no. 1, pp. 87-111, 1989.

[16] Yeh, T., Patt, Y.N., "A Comparison of Dynamic Branch Predictors that use two levels of Branch History," Proc. 20th Symposium on Computer Architecture, San Deigo, pp. 257-266, May 1993.