Branch Prediction using Advanced Neural Methods

Sunghoon Kim
Department of Mechanical Engineering
University of California, Berkeley
shkim@newton.berkeley.edu

Abstract

Among the hardware techniques, two-level adaptive branch predictors with two-bit saturating counters are acknowledged as best branch predictors. They accomplish very competitive performance at low hardware cost. However, with the rapid of evolution of superscalar processors, the more accurate predictors are desired for more correct branch prediction as one of speculation method. They will lead to higher performance of processors with no doubt. This article presents alternative new and potential neural net methods for branch prediction. The advanced applications of the neural networks more than perceptron or backpropagation are examined as alternative methods. They are radial basis networks, Elman networks and Learning vector quantization (LVQ) networks. I demonstrate that these neural methods can achieve misprediction rate comparable to the conventional two-level adaptive predictors, representatively a Gshare method, without consideration of hardware and prediction latency. I also present the effects of the history length of the global history shift register (HR) and the size of the pattern history table (PHT) on the misprediction rate for each neural method.

1. Introduction

As instruction per cycle (IPC) of the superscalar processor increases, more accurate dynamic branch predictions are increasingly required for better performance. This is because more instructions will be lost so there are more miss penalties in order to make a correction of branch prediction. Therefore, a little difference in mispredicton rate will result in big loss of instructions and loss of substantial performance of the processor.

For branch prediction to improve the performance of processors branch instructions should be detected and their branch behaviors such as their taken or not taken, or their next target PC addresses, must be correctly predicted. Moreover, the prediction of the current branch instruction must be completed in time to fetch a next instruction from the branch target address without retarding a next instruction getting into the pipeline of the processor. A classic branch predictors such as famous two-level adaptive branch predictors tried to achieve these objectives by storing the history of previously executed branch instructions into either a Global history shift register or a per-address branch history table, and a global pattern history table or a per-address PHT: the address of the branch instruction, the branch target address and information on the previous outcomes of the branch instruction. Using these history tables and registers, the prediction on future branch instructions is performed. The PC address is used as an index to access the PHTs or the branch history tables in parallel with the instruction fetch procedure. So the prediction values of that branch instruction on the fetch stage can come out by the predictor. Straightforward prediction methods of the conventional dynamic branch predictors achieve a prediction accuracy range from 80% to 95%. This accuracy turned out to be adequate for scalar processors, but is generally considered as inadequate for superscalar processors.

In the early 90s, the demand for higher accurate branch prediction urged to develop two-level adaptive branch prediction [2]. These two-level predictors make use of two levels of branch information to make a branch prediction. The first level has branch HR which is either global or per-address. It keeps recording the branch outcome, i.e. either taken or not taken, of last branch instructions committed. How many outcomes are stored depends on it history length. In the global HR, it records all consecutive outcomes of the last branch instruction stream no matter what PC address they have, while in the per-address history table, it records the last consecutive outcomes of each branch instruction. How each branch is distributed over the per-address history table depends on how many indices are available. The second level of the predictor is the PHT that records the occurrence of the first level predictor. If its occurrence tends to be taken, it predicts the branch outcome as taken, or if the occurrence tends to be not taken, it predicts as not taken. The popular PHT in the second level consists of two-bit saturating counter that is indexed by the HR of the first level. There are two-bit counters as many as the entries of the PHT. If one of these counter is referred by the HR, it is used to predict the outcome of a branch instruction as taken when the most significant bit is one, or not taken when zero. There are two distinct prediction techniques have been developed. If a global HR is used, the predictor exploits correlation between the outcome of consecutive branch instructions, while if a local HR, i.e. per-address history table, is used, then it exploits correlation between the consecutive outcome of the same branch instruction. The performances of these two methods are dependent on the combination of branch instructions stream.

In this paper, alternative and potential methods in the neural networks are explored and tested to explore their usefulness in the prediction. As fundamental methods, the perceptron and the backpropagation that have been exploited in many paper, are also implemented. In addition, the advanced neural methods such as radial basis networks, Elman networks and Learning vector quantization (LVQ) network are explored and implemented to see their possibility and potentiality as branch predictors in the case when complicated computations, such as floating point multiplication or addition, are feasible. Simulations are performed according to the hardware budget, the length of HR to understand the relation with the neural methods and how they are improved. Through trace driven simulation, it is demonstrated that some of advanced neural predictors can achieve success rate that are comparable to conventional two-level adaptive predictors when more accurate branch predictors are required.

2. Previous Work

The paper by Calder [3] has

discussed the application of neural networks as the problem of branch prediction. He is concerned entirely with static or compile-time branch prediction so his predictions are based on information about a program's structure determined by a compiler. For example, a branch successor path that leads out of a loop is less likely to be followed than a path that remains within the loop. Thus, he achieve a misprediction rate of only 20% using a neural network, that is remarkably low for static branch prediction. But since his predictions were performed at compile time, he was unable to make use of the dynamic branch histories for his neural networks. However, his static branch prediction can be useful for performing static compiler optimizations and providing extra information to dynamic branch predictors.[5]

As dynamic predictors using neural methods, the paper by Jimenez and Lin [4] introduced the basic perceptron which is the simplest version of neural methods. It uses only global history information and compared the perceptron with two dynamic global predictors, gshare and bi-mode. The paper by Vintan and Iridon[6] also suggested the learning vector quantization (LVQ) as an another neural method for dynamic branch prediction. The neural method using learning vector quantization is also implemented in this paper. It was found that the LVQ prediction was about as accurate as a table-based branch predictor, but unfortunately, it is unable to be implemented at high-speed because it required complex computations including floating point numbers.

Even thought it is obvious that the perceptron method compared with other methods is promising and can be an alternative of the two-level adaptive predictors since it has accuracy superior to them and can be implemented efficiently in the hardware. That is because it only requires integer additions instead of floating point computations. However, it is also necessary to explore other neural methods even though they are hard to be implemented in hardware due to their complexity in computation. In the case when the hardware budget is insignificant thankfully due to tremendous transistors and complex computation involving floating point numbers are affordable in the future processors, the miss prediction rate of a branch predictor is only remained to be a critical factor in the performance of the branch predictor

3. Branch Prediction Models

There are six models presented: Gshare, Perceptron, Backpropagation, Radial basis network, Elman network and LVQ network. Except the Gshare predictor, the others are all neural methods. The Gshare predictor was implemented for comparison reason with the other methods. In all cases, the prediction is based on two inputs: the branch PCs and the actual outcome of them either taken or not taken. And also there are only a global HR and a global PHT in all cases.

It is assumed that all predictions by the predictors are made during the instruction fetch stage in the processor pipeline. When branch instructions are detected, their prediction outcome is ready at the end of the fetch stage. Thus, no matter how complex the computation is, the prediction is assumed to be completed before its output is required. The following are descriptions of each prediction method.

3.1 Gshare predictor

As one of powerful two-level branch predictors, the Gshare predictor is used in the simulation for guiding a criterion of performance improvement. The PC address of a branch instruction to be predicted is XOR-ed with a global history shift register and its result value is used as an index to a PHT. Thus, the size of the PHT is determined by the number of bits in the global HR. If there are k bits in it, there must be 2^k entries in the PHT. This predictor was designed from the GAg (Global HR and Global pattern table) predictor to overcome the destructive aliasing by randomizing the index by XORing the global HR with the branch PC address. Each entry in the PHT has two bit saturating counter. The outcome of prediction depends on the most significant bit of the counter. If the most significant bit of the counter indexed is one, the prediction of the branch is taken. If not, it is not taken. Figure 1 shows the model of Gshare.

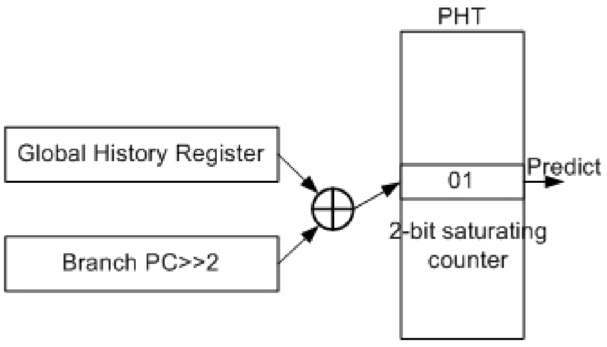


Figure 1. Gshare Model

3.2 Perceptron Predictor

The perceptron is the simplest version of the neural methods. And for the even simplest perceptron, a single-layer perceptron that consists of several input units by weighted edges to one output unit. A perceptron learns a target Boolean function $t(x_1,\ldots,x_n)$ of n inputs. In this case, the x_i are the bits of a global HR, and the target function predicts whether a particular branch will be taken or not. Thus, intuitively, a perceptron keeps track of positive and negative correlations between branch outcomes in the global HR and the branch being predicted. [5]

Figure 2 shows a graphical model of the general global neural net model that consists of the global HR and the global PHT. In this paper, all neural models have the same structure as Figure 2 and only differences between them are their neural net models to compute prediction outcome.

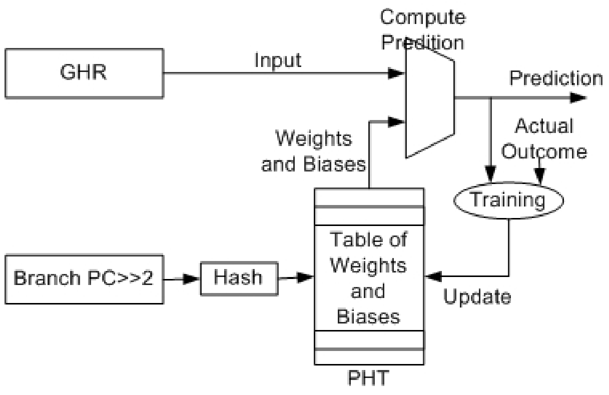


Figure 2. Block diagram of General global neural net model

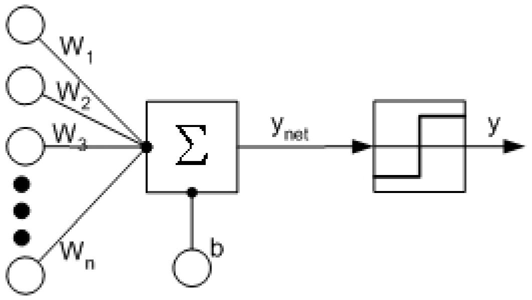


Figure 3. A single layer perceptron model

Figure 3 shows a single layer perceptron model used in this paper. Given a perceptron with weights $w_1, w_2, \ldots, w_n$, a bias b and an input pattern $x_1, x_2, \ldots, x_n$, then the net output y_{net} of the perceptron is computed as the weighted sum

$$y_{net} = b + \sum_{i=1}^{n} w_i x_i$$

$$y = sgn(y_{net})$$

The inputs to the perceptron are bipolar, i.e. either -1 when not taken or 1 when taken. The sgn function returns the sign of a input, so when a input is negative, it returns -1 and hen nonnegative, it returns 1. Thus, when the actual prediction output y is 1, the prediction of a branch is taken. If not, it is not taken. Since the weights and a bias of the perceptron are 8-bit integer values and the inputs are either -1 or 1, so only integer addition functional units are required to compute the output. This is the most attractive benefit that the other neural methods don't have since they need the floating point functional units which takes much longer time to compute than integer units.

To train the perceptron, the following algorithm is applied after the prediction output y is calculated and the actual outcome or target t is given. The target t is also bipolar, 1 if the branch was taken, or -1 if not taken. And there is a predefined value called training threshold θ that is used to provide the boundary for training. Learning weights and a bias is executed when the following conditions are satisfied: 1) if the prediction output y and the actual outcome t are different, and 2) the magnitude of the net output y_{net} is smaller than the threshold θ. Then, the weights and the bias are updated by the algorithm that increments the ith weight by one when the t agrees with x_i, and decrements it by one when the t disagrees. Thus, the weights and the bias are trained to store the correlations between the inputs and the output. The weights and the bias increase up to 127 and decrease down to -127 since they are 8 bits.

3.3 Backpropagation prediction

The prediction method exploited in this chapter is the backpropagation networks. The inputs as in the perceptron are fed into the feedforward backpropagation networks as shown in Figure 4.

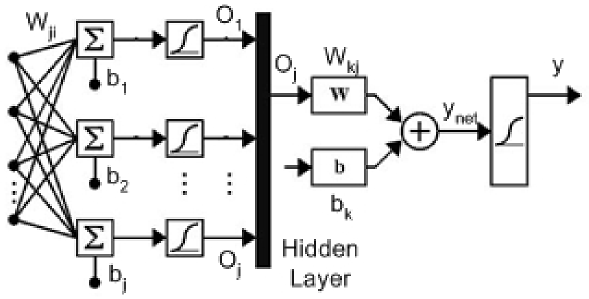


Figure 4. Two layer backpropagation model

The inputs to the net are coded in the binary way using 0 if not taken and 1 if taken. Each input is weighted with its appropriate weight W_{ji} and the sum of the weighted inputs and the bias forms the input to the transfer function. The log-sigmoid transfer function is used in this case. Then the outputs of the transfer function are fed into the hidden layer as inputs. Through the same procedure from the hidden layer to the output layer, the output prediction value is derived. The log-sigmoid transfer function used and the relationship between inputs and outputs are as follows

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

$$O_{jnet} = \sum_{i=1}^{n} W_{ji} x_i + b_j$$

$$O_j = f(O_{jnet})$$

$$y_{net} = \sum_{j=1}^{m} W_{kj} O_j + b_k$$

$$y = f(y_{net})$$

The dimensions of n, m are the size of input vectors and the size of hidden layer, or the number of neurons in the hidden layer respectively. In this case, k is always one since there is one output neuron. Thus, the prediction output y is used to predict in such a way that if it is higher than 0.5, the branch prediction is taken, otherwise it's not taken. This is because the network uses binary inputs.

Likewise, this feedforward backprop-agation network has one hidden layer of sigmoid neurons followed by an output layer of the same sigmoid neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors.

To train the network, the gradient descent algorithm is used. This simplest implementation of backpropagation training algorithm updates the network weights and biases in the direction in which the performance function decreases most rapidly - the negative of the gradient. More detail about the training algorithm is skipped.

3.4 Radial Basis network prediction

The first advanced neural networks exploited in this paper is the radial basis network as shown in Figure 5

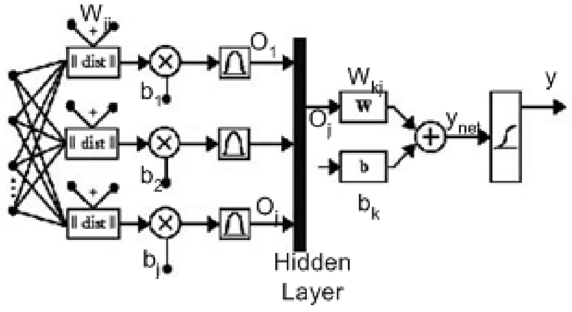


Figure 5. Radial basis network model

Radial basis networks consist of two layers: a hidden radial basis layer of m neurons and an output linear layer of one neuron. The $\|dist\|$, i.e. the distance function box, in this figure accepts the input vector (x_1, x_2, …, x_n) and the input weight matrix W_{ji}, and produces a vector having m elements. The elements are the distances between the input vector and vectors W_{ji} formed from the rows of the input weight matrix. The bias vector b_j and the output of $\|dist\|$ are combined and produce the input to the radial basis function in the first layer. This radial basis function is followed

$$f(x) = e^{-x^2}$$

The radial basis function has a maximum value of 1 when its input is 0 and as the distance between the weight vector and the input vector decreases, the output value will increase. Therefore, the radial basis networks can be used as a detector to produce 1 whenever the input vector is identical with the weight vector and produce 0 when the input vector is quite different from the weight vector. The bias *b* is used to adjust the sensitivity of the neuron

3.5 Elman network prediction

The second advanced neural network is the Elman network, one of recurrent networks as shown in Figure 6.

The Elman network is commonly a two-layer network which has feedback form the first-layer output to the first layer input. This recurrent feedback gives the Elman networks to detect and generate time-varying patterns. As like the backpropagation model, the input to the transfer function consists of a sum of the weight vector multiplied by the input vector and the bias. In addition in the Elman network, the previous outputs of the first-layer multiplied by the weight matrix is added to the input to the transfer function.

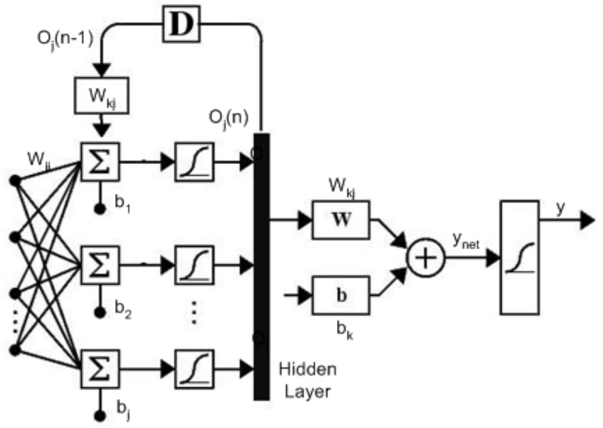


Figure 6. Elman network model

Thus, the Elman network is different from conventional two-layer neural networks because the first layer has a recurrent connection. The values stored at the previous time step is used to compute for the values at the current time step. The one requirement in the Elman networks is that there have to be enough neurons in the hidden layer. More hidden neurons are required as the function being fit increases its complexity. One benefit of this method is that it is able to learn temporal patterns as well as spatial patterns since the network stores information for the future reference.

3.6 LVQ network prediction

A LVQ network in this paper is shown in Figure 7. It is consisted of a first competitive layer which has a competitive block and a second layer of the sigmoid function. The distance blocks are used as the Radial basis network and this particular competitive layer automatically is trained to classify input vectors. This classification is dependent on the output of the distance function that computes the distance between the weight vector and the input vector. The competitive block compares all inputs to it and chooses a winning input which has the maximum value. The block gives one to the winning neuron and zeros to the other neurons. In this way, the competitive layer only detects the neuron whose weight vector has the closet distance to the input vector. If two input vectors are very similar, the competitive layer classifies them in the same class.

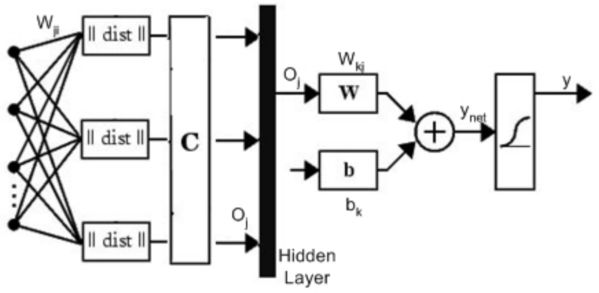


Figure 7. LVQ network model

4. Simulation results

The simulations used the trace of branch instructions of some SPEC2000 benchmarks simulated by the simplescalar tool. Total 100,000,000 instructions were executed and as a result, 10,000,000~ 20,000,000 branch instructions were executed on average for each benchmark.

Six prediction model discussed above were simulated: Gshare, perceptron, backpropagation, Radial basis network, Elman network and LVQ network models. Basic strategies for the simulation are based on 1) same hardware budget, 2) various size of the PHT and 3) various length of the GHR. The short descriptions about benchmarks are followed:

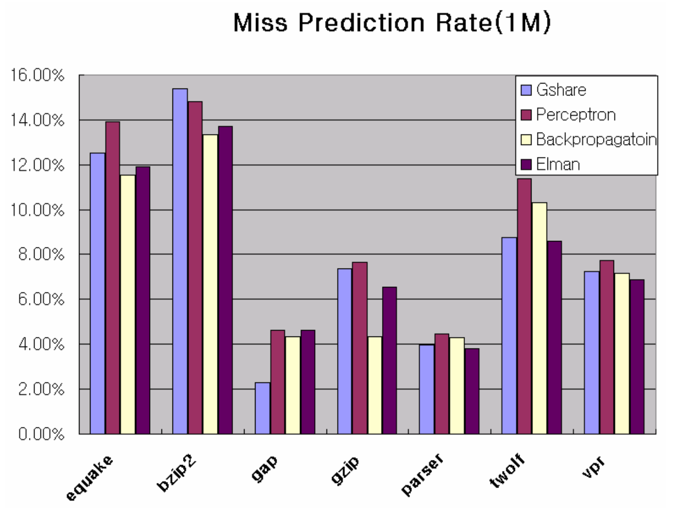
- euqake – floating point benchmark, Simulation of seismic wave propagation in large basins
- bzip2 – integer benchmark, compression

- gap – integer benchmark, Group theory, interpreter
- gzip – integer benchmark. Compression
- parser – integer benchmark, Word processing
- twolf – integer benchmark, Computer Aided Design
- vpr – integer benchmark, Integrated Circuit Computer-Aided Design Program

4.1 Miss prediction rate on the same hardware budget

Figure 8 shows the miss prediction rates of the different predictors on the different SPEC2000 trace with budgets of 1MB and 256KB. The results of the Radial basis network and the LQV networks were omitted in these figures due to their ill-performance. The results from them will be discussed separately later.

In the figure, we found that the Gshare had better performance than the others only in the *gap* benchmark. In the other benchmarks, the backpropagation is the best in *equake*, *bzip2*, and *gzip*, while the Elman network is the best in *parser*, *twolf* and *vpr*



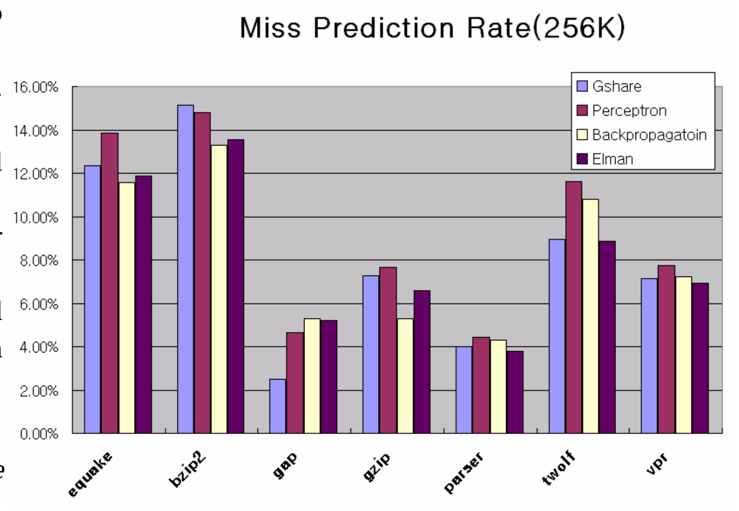
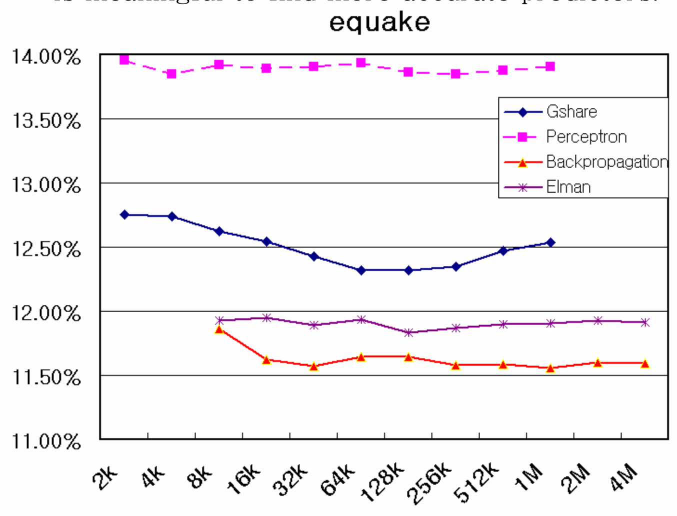


Figure 8. Miss prediction rate of different predictors with 1MB and 256KB hardware budget

These results provide evidence that the complicated neural method such as the Backpropagation and Elman network can achieve better performance than two-level predictors.

4.1 Impact of PHT size

Intuitively, the big size of the PHT may lead to the more accurate predictor. Since transistor densities increase dramatically, the exploration of miss prediction rate as hardware budgets increases is meaningful to find more accurate predictors.



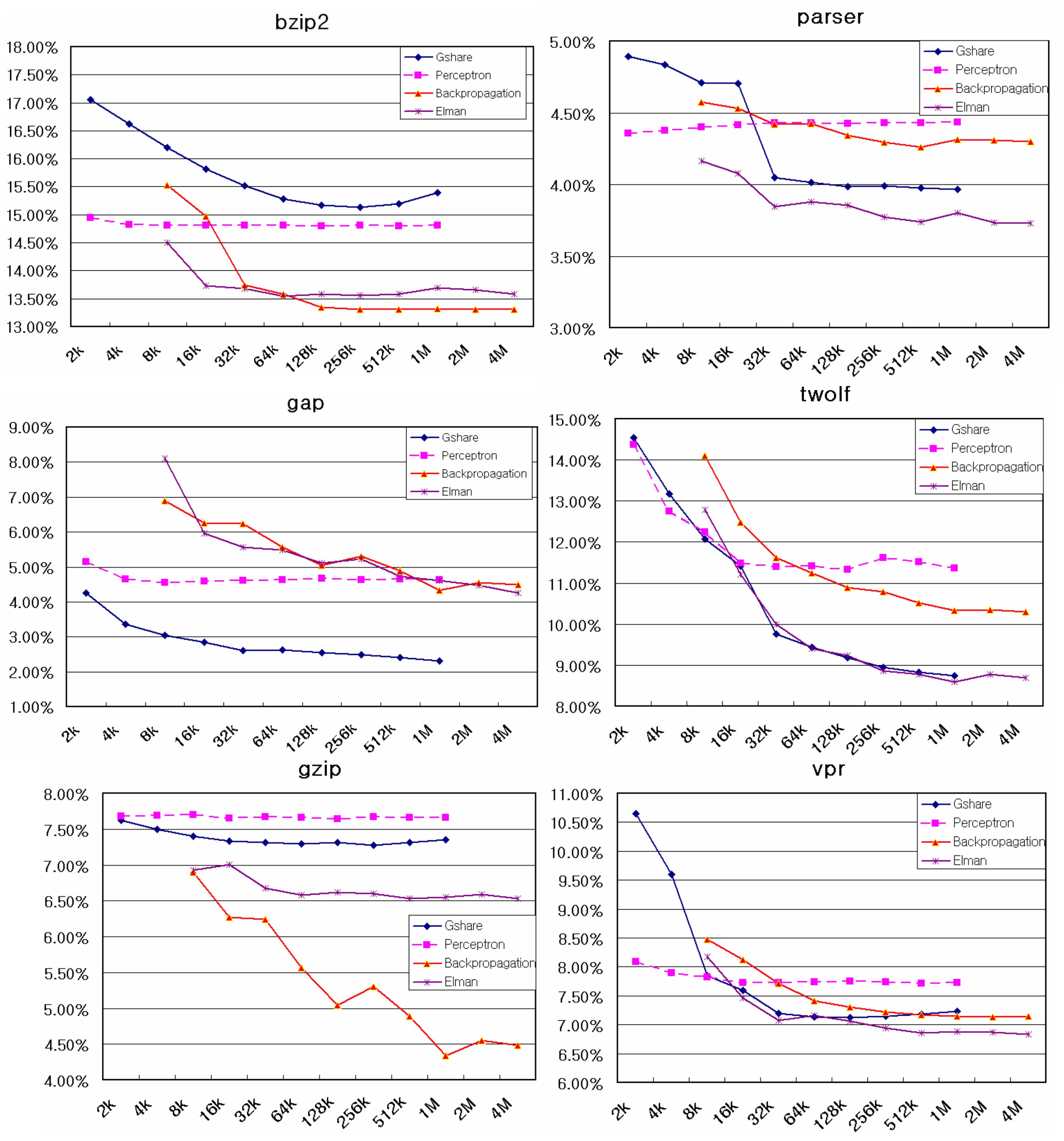


Figure 9. Impact of PHT size increase for each benchmark

From the above figures, it is concluded that generally, the miss prediction rate is reduced as the PHT size increase. In the simulation, the Gshare has from 8,192 entries at the minimum when 2KB budget to 4,194,304 entries at the maximum when 1MB budget. Likewise, the perceptron has from 2,048 at 2KB to 1,048,576 entries at 1MB. But the backpropagation and the Elman networks has relatively small amount of entries in the PHT since each entry has much longer columns than the Gshare and the perceptron. They have only from 8 entries at 8KB to 4,096 entries at 4MB. Therefore, the index interference problem or the aliasing problem is expected to be more critical as the number of entries decrease, especially in the backpropagation and the Elman network since they have only 8 entries PHT at the worst case. However, surprisingly this problem only degraded the performance of the miss prediction rate by 3~4% at maximum. As a result, this property can be considered as one of advantages of neural methods as if they buffered the aliasing problem.

4.2 Impact of HR length

The length of the HR is also one of factors to determine the performance of predictors. Long history register length is more likely to reduce the miss prediction rate of the predictor.

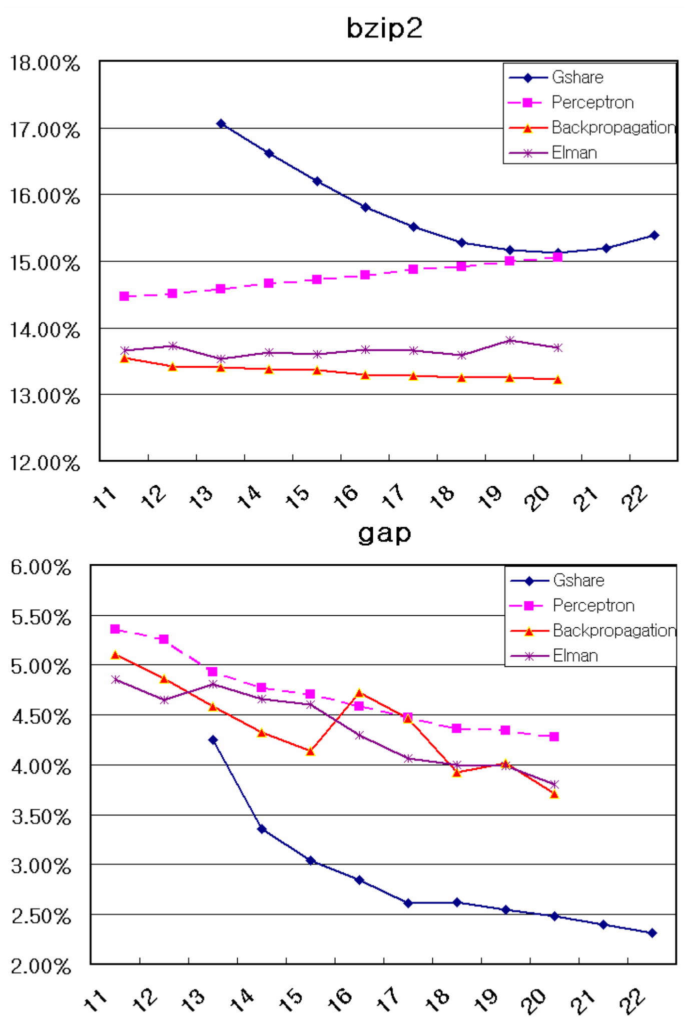


Figure 10. Impact of HR length

From the above figures, it is noticed that the HR length doesn't affect on the missprediction rate of the neural methods as much as on the Gshare. This is because that more information are stored in the PHT of the neural methods than in the PHT of the Gshare. So even though the information is lost in the HR, the neural nets are not influenced as the Gshare.

4.3 Performance of the Radial Basis network and the LVQ network

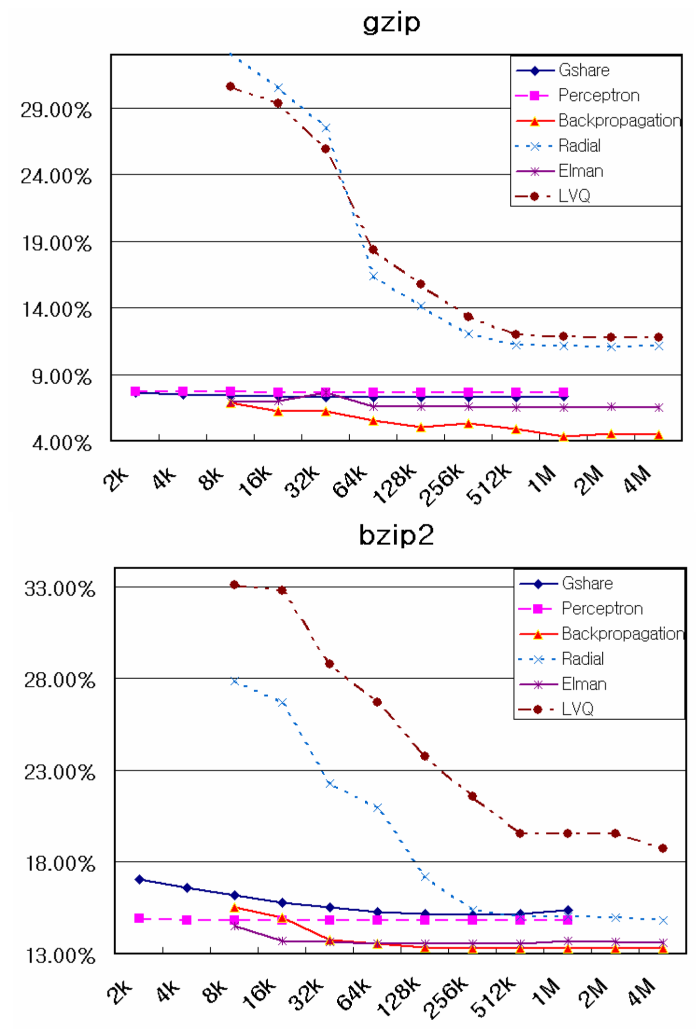


Figure 11. Performance of the Radial basis network and the LVQ network

The above figure shows the performance of these two networks over specific benchmarks. We can notice that they achieved much worse performance than the others. In this simulation, the fundamental training algorithm was applied to train the networks, that is, the weight vectors are updated in the way to reduce the distance to the input vector. However, from the results, further studies are required to improve the performance either on the training algorithm or the neural network architecture.

5. Conclusion

In this article, more advanced neural networks such as the Radial basis network, the Elman network and the LVQ network are exploited to determine their possibility as branch predictors. In the case where the hardware implementation for these predictors are feasible, these methods are attractive since they can achieve better performance in branch prediction than conventional two-level branch predictors. Six prediction models were presented and designed, and for each case, the simulation was implemented in the condition of 1) the same hardware budget, 2) various PHT size and 3) various HR length. At last, it is found that the neural methods achieve almost equal or better performance than the Gshare on the same hardware budget. Moreover, the aliasing problem that is the critical issue in the two-level predictors didn't degrade the performance at even extreme case such as 8-emtry PHT. Therefore, it can be concluded that the neural methods has possibility to be used as a high accurate predictor and an alternative of two-level predictors.

Further studies are required for the case of the Radial basis network and the LVQ network method. It turns out that they achieved worse performance than the other predictors, Better training algorithm or the different structure of networks may improve their performance.

References

[1] J. L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 3rh edition

[2] T. Yeh, T. and Y. N. Patt. *Two-Levles Adaptive Training Branch Prediction*, Micro-24, Albuquerque, New Mexico, November 1991, pp51-61

[3] B. Calder, D. Grunwald and D. Lindsay. *Corpus-based static Branch Prediction,* SIGPLAN Notices, June 1995, pp79-92

[4] Jimennez, D.A. and Lin, C. 2001, *Dynamic branch prediction with perceptrons*. In Proceeding of the Seventh International Symposium on High Performance Computer Architecture, 197-206

[5] Daniel A. Jimenez and Calvin Lin, Neural Methods for Dynamic Branch Prediction, ACM Transactions on Computer Systems, Vol. 20, No. 4 November 2002, Pages 369-397

[6] Vintan, L. and Iridon, M. 1999. *Towards a high performance neural branch predictor*. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), vol. 2. 868-873

[7] Rupak Majumdar and Dror Weitz, *Branch Prediction using Neural Nets*, CS252 project Fall 2000