Dynamic Branch Prediction using Neural Networks

Gordon Steven[1], Rubén Anguera[1], Colin Egan[1], Fleur Steven[1] and Lucian Vintan[2]
[1.] University of Hertfordshire, Hatfield, UK.
[2.] University "Lucian Blaga" of Sibiu, Sibiu, Romania
e-mail: G.B.Steven@herts.ac.uk

Abstract

Dynamic branch prediction in high-performance processors is a specific instance of a general Time Series Prediction problem that occurs in many areas of science. In contrast, most branch prediction research focuses on Two-Level Adaptive Branch Prediction techniques, a very specific solution to the branch prediction problem. An alternative approach is to look to other application areas and fields for novel solutions to the problem. In this paper, we examine the application of neural networks to dynamic branch prediction. Two neural networks are considered: a Learning Vector Quantisation (LVQ) Network and a Backpropagation Network. We demonstrate that a neural predictor can achieve misprediction rates comparable to conventional Two-level Adaptive Predictors and suggest that neural predictors merit further investigation.

Key Words

Neural Branch Prediction, Two-level Adaptive Branch Prediction, Backpropagation

1. Introduction

As the average instruction issue rate in multiple-instruction-issue (MII) processors increases, accurate dynamic branch prediction becomes increasingly important. Very high prediction accuracy is required because an increasing number of instructions are lost before a branch misprediction can be corrected. As a result even a misprediction rate of a few percent involves a substantial performance loss.

If branch prediction is to improve performance, branches must be detected within the dynamic instruction stream, and both the direction taken by each branch and the branch target address must be correctly predicted. Furthermore, all of the above must be completed in time to fetch instructions from the branch target address without interrupting the flow of new instructions to the processor pipeline. A classic Branch Target Cache (BTC) [1] achieves these objectives by holding the following information for previously executed branches: the address of the branch instruction, the branch target address and information on the previous outcomes of the branch. Branches are then predicted by using the PC address to access the BTC in parallel with the instruction fetch process. As a result each branch is predicted while the branch instruction itself is being fetched from the instruction cache. Whenever a branch is detected and predicted as taken, the appropriate branch target is available at the end of the instruction fetch cycle, and instructions can be fetched from the branch target in the next cycle. Straightforward prediction mechanisms based on the previous history of each branch give a prediction accuracy of around 80 to 95% [1]. This success rate proved adequate for scalar processors, but is generally regarded as inadequate for superscalar designs.

The requirement for higher branch prediction accuracy in superscalar systems and the availability of additional silicon area led to a dramatic breakthrough in the early 90s with the development of Two-Level Adaptive Branch Prediction [2, 3]. Two-Level Adaptive Branch Predictors use two levels of branch history information to make a branch prediction. The first level consists of a History Register (HR) that records the outcome of the last k branches encountered. The HR may be a single global register, HRg, that records the outcome of last k branches executed in the dynamic instruction stream or one of multiple local history registers, HRl, that records the last k outcomes of each branch. The second level of the predictor known as the Pattern History Table (PHT) records the behaviour of a branch during previous occurrences of the first level predictor. The PHT consists of an array of two bit saturating counters that is indexed by the HR to obtain the prediction. With a k-bit HR, 2^k entries are therefore required if a global PHT is provided, or many times this number if separate HRs and therefore PHTs are provided for each branch.

Although all the new predictors are called Two-level Adaptive Predictors, this is misleading. Two distinct prediction techniques have in fact been developed. If a global history register is used, the predictor exploits correlation between the outcome of a branch and the outcome of neighbouring branches that are executed immediately prior to the branch. If a local history register is used, the predictor exploits correlation between the outcome of a branch and previous outcomes of the same branch. A global two-level predictor is more

0-7695-1239-9/01 $10.00 © 2001 IEEE

accurate than a classic BTC because it exploits new information. In contrast, a local two-level branch predictor is more accurate because it keeps a record of the next outcome in a time series, whereas a BTC records the most frequent outcome of each branch.

In complete contrast to earlier work, this paper explores the possibility of using neural networks to dynamically predict branch outcomes. Conventional two-level branch predictors rely on one of only two correlation mechanisms. One of our main research objectives is to use neural networks to identify new correlations that can be exploited by branch predictors. We also wish to determine whether more accurate branch prediction is possible and to gain a greater understanding of the underlying prediction mechanisms. Finally, we hope to design and evaluate hardware implementations of simplified neural branch predictors. Alternatively, our research may lead to the design of more effective two-level branch predictors.

In this first paper on dynamic neural branch prediction, we explore the suitability of two neural networks, a Learning Vector Quantisation Network (LVQ) and a Backpropagation Network, for branch prediction. Through trace driven simulation, we demonstrate that neural predictors can achieve success rates that are comparable to conventional two-level adaptive predictors.

2. Previous Work

As far as we are aware, only one paper by Calder [4] has discussed the application of neural networks to the problem of branch prediction. Calder is concerned entirely with static or compile-time branch prediction. His predictions are therefore based on information about a program's structure that can be readily determined by a compiler. For example, a branch successor path that leads out of a loop or function is less likely to be followed than a path that remains within the loop or function. Using a neural network, Calder achieves a misprediction rate of only 20%, remarkably low for static branch prediction. Since Calder's predictions were performed at compile time, he was unable to feed the dynamic branch histories used by two-level predictors into his neural networks. As a result, perhaps the most useful contribution of his paper is to suggest a wide range of alternative inputs that might correlate with branch outcomes and which might therefore be usefully added to dynamic predictors.

Since most recent research on branch prediction has concentrated on two-level adaptive techniques [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], it is useful to explore some of the drawbacks of two-level predictors. The main disadvantages can be found in the following areas:

- High cost of PHT
- Branch interference
- Slow Initialisation
- "Hard-to-Predict" Branches

With a global predictor using a single Pattern History Table (PHT) 2^k entries are required, where k is the number of bits in HRg. This is the GAg predictor in Patt's classification system [11]. To achieve a 3% misprediction rate with the Spec benchmarks an 18-bit HRg is required [11] giving a PHT with 2^{18} or 256K entries. Moreover, if a separate PHT is used for each branch, as in Patt's GAp configuration, a far greater area must be devoted to the PHT. For example if HRl is 12 bits long and 16 bits are used to distinguish each branch, a total of 2^{28} entries is required. Similar high storage requirements apply to local branch predictors - PAg and PAp in Patt's system - even though fewer local history register bits are generally required to achieve a given success rate.

The high implementation cost of conventional two-level predictors has had a subtle, but important, impact on branch prediction research; it has discouraged any developments that increase the size of the PHT. Perhaps the most obvious example is that researchers are deterred from attempting to extract additional prediction accuracy from very long HRs. Similarly, researchers are discouraged from describing the program path leading to each branch more accurately by recording the full path information [14], and from combining global and local history information in a single predictor. Finally, the use of additional prediction information, such as the branch direction information used in the simple BTFNT (Backwards Taken, Forward Not Taken) heuristic, is discouraged.

The high cost of a conventional PHT suggests that alternative configurations should be considered. One possibility is to replace the PHT with a Prediction Cache [15, 16]. Although a tag field must be added to each PHT entry, the very large size of conventional PHTs suggests that the total number of entries and therefore the total cost of the PHT could be significantly reduced. The danger with a Prediction Cache is that cache misses will increase the misprediction rate. The impact of prediction cache misses can, however, be minimised by restoring a two-bit prediction counter to each entry of the conventional BTC, which is still required in all two-level predictors to furnish the target address for each branch. These BTC counters can then be used to provide a default prediction whenever there is a miss in the prediction cache [15]. Alternatively, an entirely different approach, such as the neural branch predictors introduced in this paper, can be investigated.

The second problem, branch interference, is a direct result of excessive PHT sizes. Because of the high cost of PHTs, researchers are reluctant to provide a separate PHT array for each branch. Instead, each PHT is shared, either between all branches in the GAg or PAg configurations, or between a set of branches in the GAs

and PAs configurations. Unfortunately, prediction information is now also shared between several branches, leading to interference [9]. The Gshare Predictor [8, 17] attempts to minimise this interference by hashing the PC and HR fields before accessing the PHT. The objective is to spread accesses more evenly throughout the PHT. Alternatively, the Bimodal Predictor [7] uses twin PHT arrays to decrease destructive interference between branch predictions and to maximise positive interference. Note that the problem of branch interference can be completely avoided by using a Prediction Cache.

The third problem is PHT initialisation. In the worst case, the 2^k prediction counters associated with each branch, where k is the length of the HR, must be trained before the predictor is fully effective. Even allowing for the fact that a PHT is effectively a sparse matrix with many unused entries, this situation contrasts sharply with a classic BTC that is fully initialised after one execution of each branch. The impact of PHT training can be reduced by combining a two-level predictor with a classic BTC in a hybrid predictor [5]. The BTC will then be used in preference to the two-level predictor while the latter is being initialised. Alternatively, explicit PHT counter initialisation can be combined with the simple default predictor in the BTC mentioned earlier [18].

Finally, some branches remain stubbornly hard-to-predict [19, 20]. There are two cases. The outcome of some data dependent branches is effectively random and these branches will never be accurately predicted. However, it should be possible to predict certain branches that are currently hard-to-predict more accurately by identifying new correlation mechanisms and adding them to the prediction process. We suggest that neural predictors may prove to be a useful vehicle for investigating potential new correlation mechanisms.

We have emphasised earlier that most branch prediction research is based on two closely related correlation mechanisms. Yet branch prediction is a specific example of a far more general time series prediction problem that occurs in many diverse fields of science. It is therefore surprising that there has not been more cross-fertilisation of ideas between different application areas. A notable exception is a paper by Mudge [19] that demonstrates that all two-level adaptive predictors implement special cases of the Prediction by Partial Matching [PPM] algorithm that is widely used in data compression. Mudge uses the PPM algorithm to compute a theoretical upper bound on the accuracy of branch prediction. In a later paper, Steven [18] demonstrates that a two-level predictor can be extended to implement a simplified PPM algorithm with a resultant reduction in the misprediction rate. Time series prediction is also an important research topic in neural networks. It therefore appears natural to look to neural networks for a further cross-fertilisation of ideas.

In this paper, we apply neural networks to dynamic branch prediction. Our objective is to demonstrate that neural networks can achieve the same prediction accuracy as a conventional Two-level Adaptive Predictor. In this initial study, we therefore restrict our neural network inputs to using the same dynamic HR information as a conventional two-level predictor.

3. Branch Prediction Models

We first briefly consider the performance of a simple LVQ (Learning Vector Quantisation) neural predictor. We then compare the performance of conventional two-level adaptive predictors with a Neural Predictor using a Backpropagation Network. In both cases the prediction process is based on two inputs: the branch PCs least significant bits and the history of the k previous branches

We assume that all predictions are being made during the Instruction Fetch stage of the processor pipeline. All our predictors therefore operate in parallel with a classic BTC that detects branches and furnishes the branch target address. The actual prediction is generated by either a neural network or a two-level predictor. Nevertheless, a miss in a BTC always results in a default prediction of not taken, irrespective of the prediction delivered by the predictor. A 1K four-way set associative BTC is used throughout the paper.

3.1. An LVQ Neural Predictor

The first neural network we examined was an LVQ [21] model. Our objective was to determine whether respectable success rates could be delivered by a simple LVQ network that was dynamically trained after each branch prediction.

The LVQ predictor contains two "codebook" vectors: the first vector, Vt, is associated with the branch taken event and the second, Vnt, with the not taken event. Vt is initialised to all ones and Vnt to all zeros. During the prediction process, the input parameters of the predictor are concatenated to form a single input vector, X. Hamming distances are then computed between X and the two codebook vectors.

$$HD = \sum_i (X_i - V_i)^2$$

The vector with the smallest Hamming distance is defined as the winning vector, Vw, and is used to predict the branch. A win for Vt therefore indicates "predict taken", while a win for Vnt indicates "predict not taken". When the branch outcome is determined, the codebook vector Vw that was used to make the prediction is then adjusted as follows:

$$Vw(t+1) = Vw(t) +/- a(t)[X(t) - Vw(t)]$$

To reinforce correct predictions, the vector is incremented whenever a prediction was correct and

decremented otherwise. The factor a(t) represents the learning factor and is usually set to a small constant less than 0.1. In contrast, the losing vector is unchanged. The neural predictor will therefore be trained continuously as each branch is encountered. It will also be adaptive since the codebook vectors will always tend to reflect the outcome of the branches most recently encountered.

3.2. Branch Prediction using a Backpropagation Neural Network Predictor

Our second neural network is a Backpropagation Neural Network [22]. The prediction information is fed into a backpropagation net (Fig. 1) which predicts the outcome of each branch. Later when the outcome of the branch is known, the error in the prediction is backpropagated through the neural network using the classic backpropagation algorithm.

Four different networks have been developed: two of them use global history information while the other two use local information. The two versions of each arise because the inputs to the net are coded in two different ways: *binary* using 0 and 1 for not taken and taken branches respectively, and *bipolar*, using -1 and 1. To exploit these different input encodings, two different activation functions are also required, a sigmoidal function for binary inputs and a bipolar sigmoidal function for bipolar inputs:

Sigmoidal function: $1/(1 + e^{-\beta x})$

Bipolar Sigmoidal function: $2/(1 + e^{-\beta x}) - 1$

The factor β controls the degree of linearity in the two activation function. In particular, as β approaches infinity, the functions become step functions, the form of the activation function used in MLP (Multi-Layer Perceptron) networks.

When predicting a branch using binary inputs, a value higher than 0.5 on the output cell is considered to be a taken prediction, whereas any value lower than 0.5 is a not taken prediction. In the bipolar case, positive values indicate a taken prediction and negative values not taken. The network is not initially trained, so random values are chosen for the weights between [-2/X, 2/X], where X corresponds to the number of inputs to the net. This selection of weights guarantees that both the weights and their average value will be close to zero and that the net is not biased initially towards either taken or not taken.

4. Trace Driven Simulation Results

4.1. Simulation Environment

Our simulations used the Stanford integer benchmark suite, a collection of eight C programs designed to be representative of non-numeric code, while at the same time being compact. The benchmarks are computationally intensive, with an average dynamic instruction count of 273,000. About 18% of the instructions are branches of which around 76% are taken. Some of the branches in these benchmarks are known to be particularly difficult to predict; see for example Mudge's detailed analysis [19] of the branches in *quicksort*.

The benchmarks were compiled using a C compiler developed at the University of Hertfordshire for the HSA (Hatfield Superscalar Architecture) [23]. Instruction traces were then obtained using the HSA instruction-level simulator, with each trace entry providing information on the branch address, branch type and target address. These traces were used to drive a series of trace-driven branch predictors. The trace-driven simulators are highly configurable, the most important parameter being the number of HR bits. As output, the simulators generate the overall prediction accuracy, the number of incorrect target addresses and other useful statistics.

4.2. Conventional Two-level Predictors

For comparative purposes, we first simulated three global predictors, a GAg predictor, a GAs predictors with 16 PHTs and a GAp predictor (Fig. 2). The average misprediction rate initially falls steadily as a function of global HR length, before flattening out at a misprediction rate of around 9.5%. In general, there is no benefit in increasing the HR length beyond 16 bits for the GAg predictor and 14 bits for the GAs/GAp predictors. Beyond this point there is either no significant benefit from new correlations or any benefit is negated by the additional number of initialisations required in the PHTs.

We also simulated three local predictors: a PAg, a PAs and a PAp predictor (Fig. 3). The local predictors achieve misprediction rates of around 7.5%, significantly better than the global predictors. The best performance is of 7.48% is achieved with a PAp predictor and a 28-bit HR. This improvement is largely achieved because local predictors, unlike their global counterparts, continue to benefit from additional HR bits. However, with both global and local two-level predictors the accuracy does not improve smoothly as a function of HR length.

4.3. An LVQ Branch Predictor

Three LVQ predictors were considered with the following inputs:

- PC + HRl
- PC + HRg
- PC + HRl + HRg

The input vector for the neural network was constructed by concatenating the least significant bits of the PC with HRg and HRl as appropriate. Initially the values of the learning step a(t) were varied between 0.1

and 0.001. Eventually the value a(t) = 0.01 was standardised after it had been demonstrated that the predictor was largely insensitive to slight variations in a(t). The simulation results are presented in Fig. 4.

The global LVQ predictor achieved an average misprediction rate of 13.54%. Furthermore, only modest further benefits were realised by increasing HR beyond four bits. The global predictor was therefore unable to benefit from large amounts of HRg information. The local LVQ predictor achieved a significantly lower misprediction rate of 10.91%. However, although this figure was recorded with a 30-bit HRl, the local predictor was also unable to adapt to a large amount of history register information, and no significant improvements were observed with HR sizes greater than 6-10 bits. The superior performance of the local predictor is not entirely unexpected; there is likely to be far more positive re-enforcement of prediction information between distinct branches in a local predictor. In contrast, a global predictor must "learn" about each branch separately.

Finally, a hybrid global and local predictor, with equal numbers of HRg and HRl bits, yielded a marginal improvement on the local predictor levels, pushing the best average misprediction rate down to 10.78%.

Overall, the results of the LVQ predictor are in line with the average accuracy of 88.10% achieved by a classic BTC with these benchmarks. A simple LVQ predictor is therefore unable to compete with a conventional two-level adaptive predictor. Nevertheless, we found these first results very encouraging. An LVQ network solves a binary classification problem by attempting to find a single multi-dimensional plane that divides the input space, in this case a taken and a not taken space, into two. Although the plane can be changed dynamically as each branch executes, it appears unlikely that an entirely satisfactory solution can be found. Since our LVQ predictor nonetheless managed to equal the performance of a classic BTC, we were encouraged to develop further neural predictors.

4.4. A Backpropagation Neural Predictor

A total of four Backpropagation Neural Predictors were simulated:

- A global neural predictor with binary inputs.
- A global neural predictor with bipolar inputs.
- A local neural predictor with binary inputs.
- A local neural predictor with bipolar inputs.

The simulation results are plotted in Fig. 5 as a function of HR length. A learning rate of 0.125 was used throughout.

The global neural predictor with binary inputs (0 or 1) achieves a misprediction rate of 11.28%, which is significantly better than the LVQ predictor. However the global predictor with bipolar inputs (-1, +1) significantly improves on this figure and achieves a misprediction rate of 8.77%. Intuitively, feeding in not taken results as minus one allows the neural predictor to exploit correlations between not taken branches and the branch being predicted. In contrast, if not taken results are fed in as zero, they can have little direct result on the final outcome, since their weighted input into each intermediate neural cell must always be zero. Interestingly, the prediction accuracy also continues to improve as HR is increased, and only finally dips below 9.0 with a HR length of 26 bits.

The local predictor consistently outperforms the global predictor. With binary inputs, the best misprediction rate is 10.46%, while with bipolar inputs 8.47% is achieved. Significantly, neural prediction performance is now comparable with the performance of two-level adaptive predictors. The best global neural predictor with a misprediction rate of 8.77% is 5.2% better than the best GAs predictor, while the best local level predictor at 8.47% is 13.2% worse than the best PAs predictor.

5. Conclusions

In this study, we sought to determine whether a neural network could mimic a two-level adaptive branch predictor and achieve comparable success rates. Two types of neural predictors were simulated, an LVQ predictor and a Backpropagation Predictor. While the LVQ predictor only achieved results comparable to a traditional BTC, the Backpropagation Predictor performance was comparable to conventional two-level adaptive predictors. In the case of global predictors, the best neural predictor was marginally superior to the best two-level predictor and, in the case of local predictors, the best two-level predictor was marginally superior. These results suggest that not only can neural networks generate respectable prediction results, but in some circumstances a neural predictor may be able to exploit correlation information more effectively than a conventional predictor.

Traditionally, neural networks undergo exhaustive and often very time-consuming training before they are used. In this respect, branch prediction appears to be an unpromising application for neural networks. In branch prediction, a neural network is expected to sample the outcome of a specific branch once and to then predict the same branch when it is encountered for a second time. The most exciting result of these simulations is therefore the extent to which backpropagation neural networks are able to assimilate and benefit from large amounts of history register information with a minimum of training. The distinct drop in the bipolar backpropagation misprediction rate when 26 bits of HR are used is a good illustration of this result. Our results therefore suggest that neural networks can adapt rapidly

enough to be successfully used in dynamic branch prediction.

Neural networks provide an extremely interesting topic for future branch prediction research. The challenge is to construct composite input vectors for neural network predictors that will enable them to outperform conventional predictors. This task involves both identifying new correlation mechanisms that can be exploited by neural prediction and tailoring the input information to fully exploit the capabilities of a neural predictor.

References

[1] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2nd edition, 1996.

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

[3] S. Pan, K. So and J. T. Rahmeh. Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation, *ASPLOS-V*, Boston, October 1992, pp76-84.

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

[5] P. Chang, E. Hao and Y. N. Patt. Alternative Implementations of Hybrid Branch Predictors, *Micro-29*, Ann Arbor, Michigan, November 1995, pp252-257.

[6] M. Evers, S. J. Patel, R. S. Chappell and Y. N. Patt. An Analysis of Correlation and Predictability: What makes Two-level Branch Predictors Work, *ISCA '25*, Barcelona, Spain, June 1998, pp52-61.

[7] C. C. Lee, I. -C. K. Chen, and T. N. Mudge. The Bi-Mode Branch Predictor, *Micro-30*, Research Triangle Park, North Carolina, December 1997, pp4-13.

[8] S. McFarling. Combining Branch Predictors, WRL Technical Note, TN36, DEC, June 1993.

[9] S. Sechrest, C. Lee and T. Mudge. The Role of Adaptivity in Two-Level Branch Prediction, *Micro-29*, Ann Arbor, Michigan, November 1995, pp264-269.

[10] E. Sprangle, R. S. Chappell, M. Alsup and Y. N. Patt. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference, *ISCA'24*, Denver, Colorado, June 1997, pp284-291.

[11] T. Yeh and Y. N. Patt. Alternative Implementations of Two-Level Adaptive Branch Prediction, *ISCA-19*, Gold Coast, Australia, May 1992, pp124-134.

[12] T. Yeh and Y. N. Patt. A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution, *Micro-25*, Portland, Oregon, December 1992, pp129-139.

[13] T. Yeh and Y. N. Patt. A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History, *ISCA-20*, San Diego, May 1993, pp257-266.

[14] R. Nair. Dynamic Path-Base Branch Correlation, *Micro-28*, Ann Arbor, Michigan, November 1995, pp15-23.

[15] C. Egan. Dynamic Branch Prediction in High Performance Superscalar Processors, PhD thesis, University of Hertfordshire, August 2000.

[16] G. B. Steven, C. Egan and L. Vintan. A Cost Effective Cached Correlated Two Level Adaptive Branch Predictor, *18th IASTED International Conference in Applied Informatics*, Innsbruck, Austria, February 2000.

[17] P. Chang, E. Hao, T. Yeh and Y. N. Patt. Branch Classification: A New Mechanism for Improving Branch Predictor Performance, *Micro-27*, San Jose, California, November 1994, pp22-31.

[18] G. B. Steven, C. Egan, P. Quick, and L. Vintan. Reducing Cold Start Mispredictions in Two-level Adaptive Branch Predictors, *CSCS-12*, Bucharest, May 1999, pp145-150.

[19] T. N. Mudge, I. Chen and J. Coffey. Limits of Branch Prediction, Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, January 1996.

[20] L. N. Vintan and C. Egan. Extending Correlation in Branch Prediction Schemes, *Euromicro99*, Milan, September 1999, Vol. 1, pp441-448.

[21] S. I. Gallant. *Neural Networks and Expert Systems*, MIT Press, 1993.

[22] R. Callan. *The Essence of Neural Networks*, Prentice-Hall, 1999.

[23] G. B. Steven, D. B. Christianson, R. Collins, R. Potter and F. L. Steven. A Superscalar Architecture to Exploit Instruction Level Parallelism, *Microprocessors and Microsystems*, Vol.20, No 7, March 1997, pp391-400.

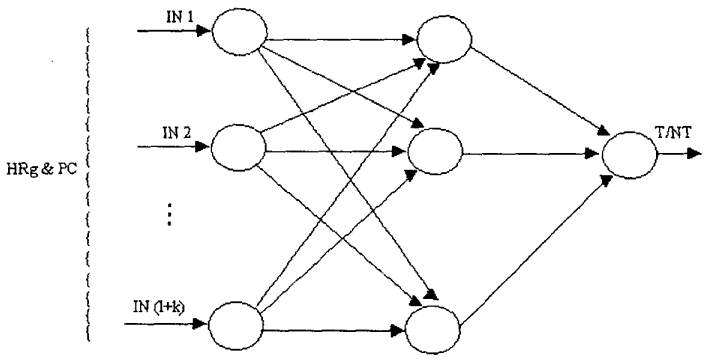

Figure 1. A Global Backpropagation Neural Predictor

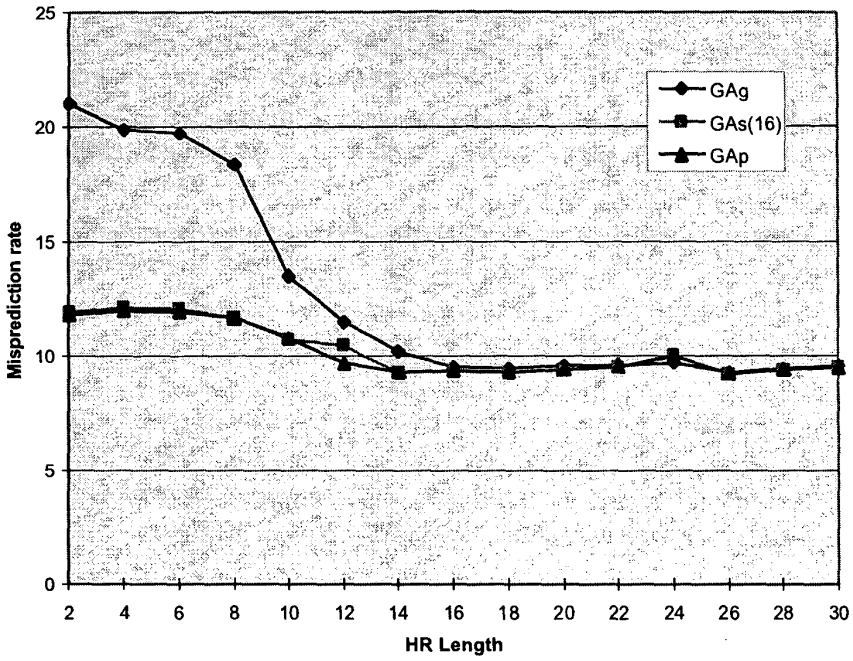

Figure 2. Global Two-Level Predictors

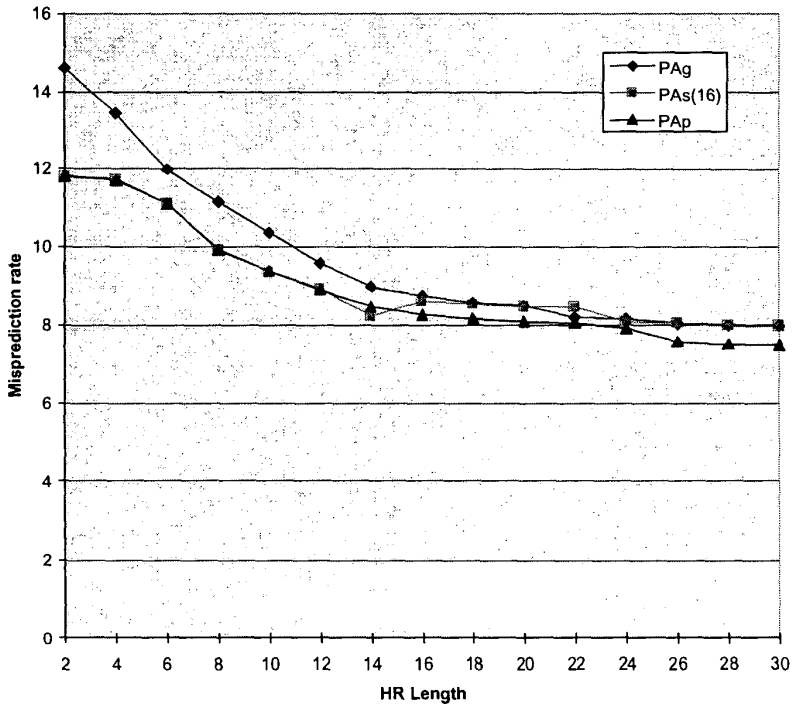

Figure 3. Local Two-Level Predictors

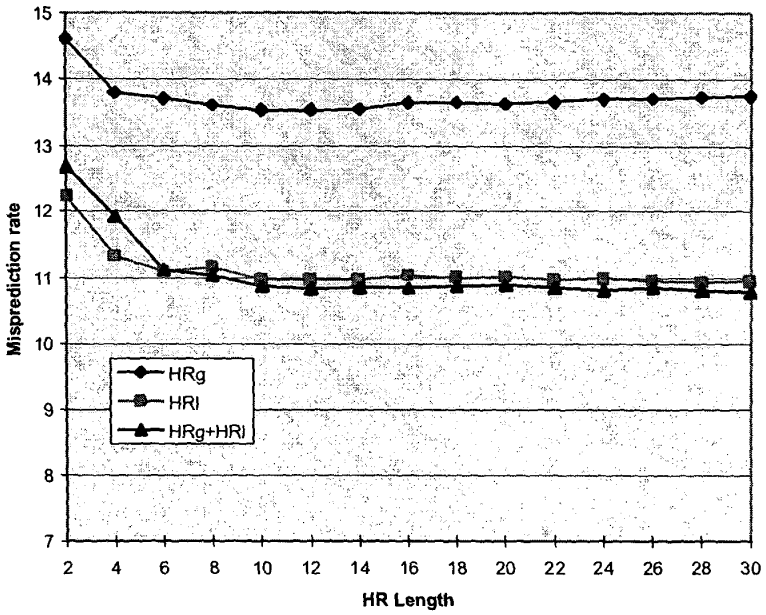

Figure 4. LVQ Branch Predictors

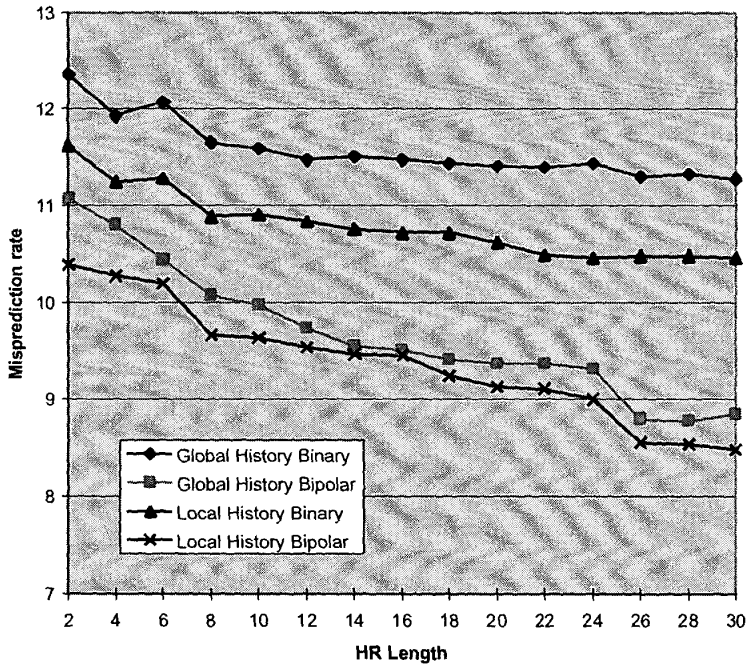

Figure 5. Backpropagation Neural Predictors