Dynamic Branch Prediction using Machine Learning Algorithms

Kedar Bellare, Pallika Kanani and Shiraj Sen
Department of Computer Science,
University of Massachusetts Amherst

May 17, 2006

1 Introduction

Modern processor architectures increasingly make use of prefetching as a way to boost Instruction Level Parallelism. If the prefetched values matches with the actual values needed at a later point of time, it reduces the latency that the processor might had to deal with. Branch prediction is one such area where accurate prediction mechanisms can lead to considerable boost in system performance.

Machine Learning algorithms have long been used to develop classifiers which learn patterns among the data for grouping them into classes. Using such algorithms for exploiting finer structure in the data seems to be a good way to address the problem of Dynamic branch prediction (DBP). However, not all conventional algorithms in machine learning can be directly applied to DBP, since they usually have a high processor time overhead.

In this report, we will be presenting various Machine Learning techniques with low memory requirements and their performance on benchmarked traces. We will also try to address the problem of *offline* learning and how that can improve the performance of the predictors.

2 Learning Branch Prediction

Dynamic Branch Prediction is a way to predict branches at run time in the hardware. If we want to learn a predictor for this, this is equivalently to learning a function f from input feature space X to output space Y, where the output space says whether *the jump was taken or not taken.*

$$\begin{aligned} f :& X \rightarrow Y \\ & X \in R^n \\ & Y \in \{-1, +1\} \end{aligned} \tag{1}$$

The features that can be considered in case of traces can be the instruction addresses, their values, the local and the global branch outcome history. The Local Branch History (LBH) applies to an instruction address or program counter and stores the history of previous branches taken from

that location. The Global Branch History (GBH) stores the number of branches which have been taken over the whole length of the trace with no reference to any particular address. The value of a particular address location can also be used as an indicator of a conditional branch, and hence a feature of the data.

Figure 1 shows a conceptual system model for a branch predictor. The source information, including branch addresses, local/global histories, along with other run-time information, is gathered when a program executes. The *information processor* extracts a subset of the source information and forms an information vector which is then fed into the predictor.

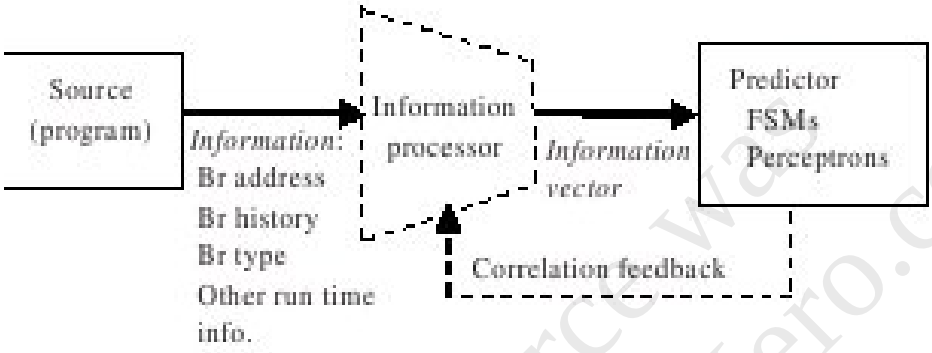

Figure 1: A conceptual system model for branch prediction

2.1 Perceptrons

Perceptrons is a simple and popular learning model which has been used for learning and classifying patterns in data. One important reason for using a perceptron predictor is that it scales linearly rather than exponentially, thereby enabling it to explore correlation from much longer information vectors. It is based on the model of neural networks in brain cells. In this model as shown in figure 2, the inputs ($x's$) are from the branch histories which are the features. Each feature is associated with a weight, w which is learned by on-line training. The output y is a dot product of $x's$ and $w's$.

$$\hat{y} = w_0 + \sum_{i=1}^{n} x_i w_i \tag{2}$$

The prediction then is based on the sign of $\hat{y}$. In the training phase this method tries to find correlations between history and outcome. This algorithm can capture patterns in data which are *linearly separable*. The training algorithm [2] proceeds as follows:

- $x_{1...n}$ is the n-bit feature vector, x_0 is 1.
- $w_{0...n}$ is the weights vector.
- y is the branch outcome.

- Learning rate was set to 1.

1. If $y = sgn(\hat{y})$, do not change weights.
2. If $y \neq sgn(\hat{y})$,

$$w_i = \begin{cases} w_i + 1, \text{ if } y = x_i \\ w_i - 1, \text{ if } y \neq x_i \end{cases} \tag{3}$$

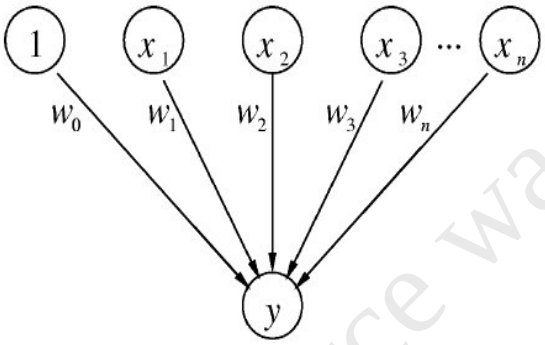

Figure 2: A Perceptron

2.2 Kernel-Based Perceptron

This algorithm is similar to the perceptron based predictor for online learning. However, in this technique we are not satisfied with obtaining a margin separating the binary classes. We are more interested in finding the optimum margin in order to increase the future prediction rates. After the prediction is made and the true label revealed, the algorithm suffers an *instantaneous loss* which reflects the degree to which the prediction was wrong [1]. At the end of each prediction, the algorithm uses the newly obtained instance-label pair to improve its prediction rule for future predictions.
If y_t is the true outcome of a prediction with x_t and w_t being the feature and weight vector, then we refer to $y_t(w_t \cdot x_t)$ as the signed margin obtained on instance t. Whenever the margin is a positive number then $sign(w_t \cdot x_t) = y_t$ with the algorithm making a correct prediction. This algorithm however doesn't stop at making a correct prediction but tries to increase the confidence of future predictions. hence, the goal is to achieve a margin of at least 1 as often as possible. If, on an instance the algorithm attains a margin less than 1 it suffers an instantaneous loss. This loss is defined by the following *hinge-loss* function,

$$l(w; (x, y)) = \begin{cases} 0 & y(w \cdot x) \geq 1 \\ 1 - y(w \cdot x) & otherwise \end{cases} \tag{4}$$

We initially start with zero weights and then use this loss function to update the weights as shown in the algorithm below.

INPUT: Parameter $C > 0$
INITIALIZE: $w_1 = (0, \ldots, 0)$
For $t = 1, 2, \ldots$

- receive instance: $x_t \in R^n$
- predict: $\hat{y}_t = \text{sign}(w_t \cdot x_t)$
- receive correct label: $y_t \in -1, +1$
- suffer loss: $l_t = \max\{0, 1 - y_t(w_t \cdot x_t)\}$
- update:
  1. set: $\tau_t = \frac{l_t}{\|x_t\|^2}$
  2. update: $w_{t+1} = w_t + \tau_t y_t x_t$

2.3 Hybrid Predictor

This predictor is similar to a Tournament predictor as it makes predictions based on multiple predictors in parallel and then chooses which predictor it like most(based on past performance). In the Tournament predictor, we usually use a combination of two predictors, one using only local history outcomes and the other using global history outcomes. In our case, we also used a combination of two predictors, on using some online learning technique like perceptrons, kernel perceptrons and the other using some standard branch prediction algorithm like Gshare or prediction based on local history. We use a two bit *predictor* to select the predictor. The state transition diagram for the hybrid predictor has four states corresponding to which predictor to use as is shown in figure 3. Based on the prediction and the actual outcome, both of the predictors are updated.

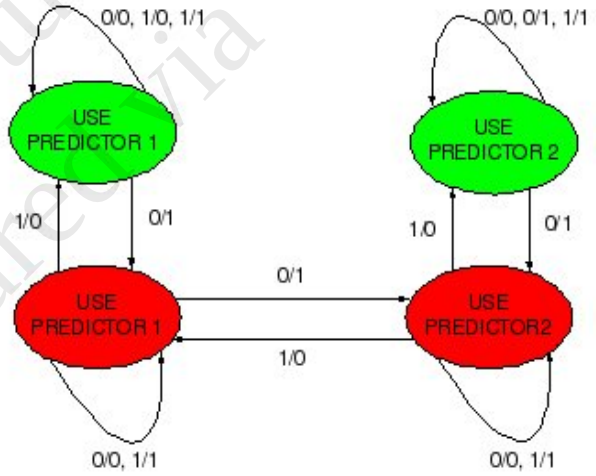

Figure 3: State Transition diagram of hybrid predictor

2.4 Adaptive Information Processing with Decision Trees

This method proposed by [3] is based on the observation that perceptron weights can be used as a quantitative measure of correlation between the current prediction and the information vector. Hence, one can adaptively re-assemble the information vector to maximize the correlation without increasing the size of the perceptron predictors.

Adaptive Information Processing can coarsely be defined from figure 4. In this, m-bits is selected from the much longer Information vector and passed to the multiplexers which assembles them and passes them on to the perceptrons. Gao et. al in their paper however hardcoded the features which an information vector can have based on the workload - Floating Point, Integer, Multimedia or Server.

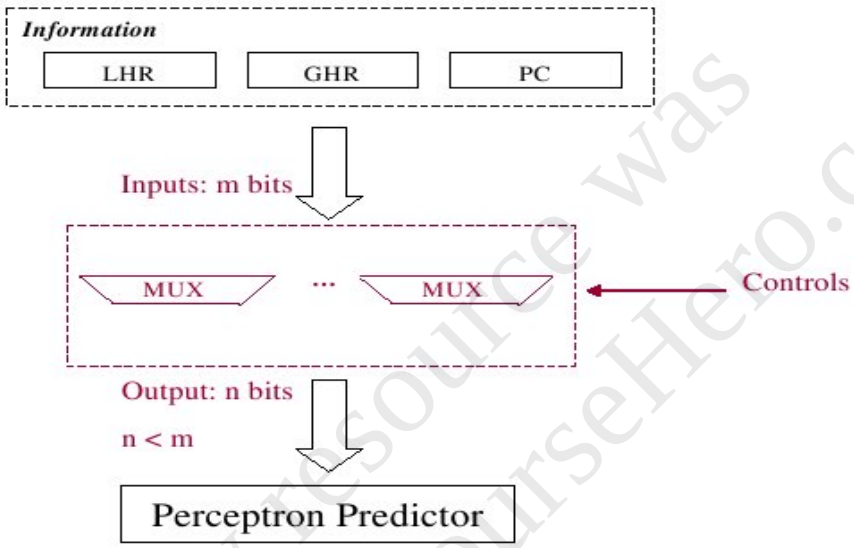

Figure 4: Adaptive Information Processing

In our approach, we learn the inputs to the multiplexer using Decision Trees. Some amount of the trace data is used for offline learning of a Decision Tree which tries to learn the best features to be used for the particular task. Once this is done, this feature vector is then fed into the perceptron for online prediction. The perceptron follows the usual online learning for re-tuning its weights.The complete predictor is shown in figure 5 where the input to the multiplexers is now learned rather than pre-determined.

3 Experimental Results

We used the dataset provided for the 1^{st} JILP Championship Branch Prediction Competition (CBP-1). The dataset consisted of 20 different traces with five each of Integer, Floating Point, Multimedia and Server. Each of the trace were approximately 30 million instances long and included both system and user activity. In our first set of results as shown in Table 1, we used online learning using the first

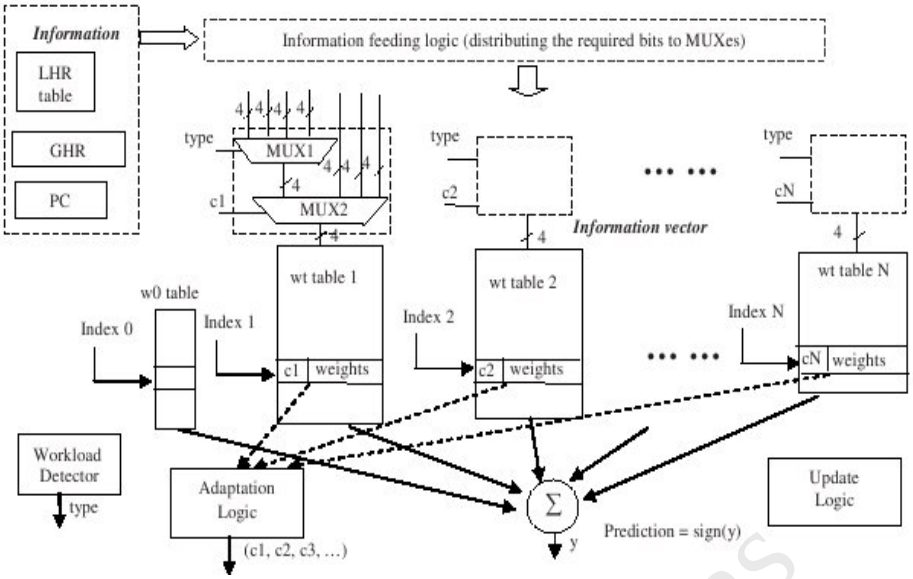

Figure 5: A MAC perceptron predictor with adaptive information processing

Trace	FP	INT	MM	SERV	AM	HM
Gshare	1.30	6.61	6.78	6.52	5.05	1.50
Local	13.62	28.24	31.07	37.72	25.80	11.96
Perceptron (using Local and Global History)	0.78	6.74	7.34	8.55	5.85	0.69
Perceptron (using Only Global History)	0.81	6.64	7.39	9.16	6	0.77
Hybrid: Perceptron+Local	3.35	16.63	16.92	35.07	15.10	3.84
Hybrid: Perceptron+Gshare (Both Updating)	1.35	7.04	7.04	7.43	5.37	1.65
Hybrid: Perceptron+Gshare (Selective Updating)	1.88	12.52	13.13	17.60	10.05	2.18
Kernel (using Only Global History)	13.26	14.19	25.77	8.51	16.63	10.43
Kernel (using Local and Global History)	13.73	16.51	27.68	9.19	18.04	9.36

Table 1: Results using Online Learning

three techniques on the dataset. In all our algorithms we used hardware within the budget of 64K inorder to simulate real hardware constraints. The results indicate the number of wrong predictions per thousand instances. The sixth and the seventh columns of the table indicate the Arithmetic and Harmonic means.

In our second set of experiments, we used offline learning to see how the prediction accuracy changes. We used part of the traces to train the classifiers and then used a testing dataset to check the results. Table 3 shows the results obtained for three such classifiers as we increases the training dataset. The number in brackets indicate the percentage of data used for training. Testing was always performed on the last 50% of the data. As can be seen from the results, there was no improvement at all with the increase in training size since the classifier learned during offline training ws not applicable to the data.

In our next set of experiments we used 10000 traces to train the decision tree and then used it for branch prediction with Adaptive Information Processing. During branch prediction, the weights of the parameters were re-tuned online depending on the results and the actual value. As can be seen from Table 3, the results are significantly better than that for Gshare.

Trace	FP	INT	MM	SERV	AM	HM
Hybrid: Perceptron+Local (20% data)	8.68	32.68	43	99.16	45.88	9.37
Hybrid: Perceptron+Local (40% data)	8.28	32.76	43.23	100.02	46.07	9.32
Hybrid: Perceptron+Local (50% data)	8.64	32.76	43.31	100.1	46.2	9.37
Perceptron (20 % data)	6.34	24.04	33.94	71.16	33.87	6.65
Perceptron (40 % data)	6.34	23.96	33.92	71.12	33.84	6.65
Perceptron (50 % data)	6.32	23.96	33.9	70.98	33.79	6.61

Table 2: Results using Offline Learning

Trace	FP	INT	MM	SERV	AM	HM
Gshare	1.30	6.61	6.78	6.52	5.05	1.50
Adaptive Information Processing with DT	0.43	3.21	4.45	3.19	2.82	0.105

Table 3: Results using Adaptive Information Processing with Decision Trees

Figure 6 shows a part of the Decision Tree learned when using a small subset of the traces for offline learning. The nodes of the tree indicate which parts of the information to be used (local history(lh), global branch history(bh) or program counter(pc)) for assembling the information vector. The number following the feature indicates which bit of the feature to be used. For eg: $lh29$ indicates 29^{th} bit of the local history.

4 Conclusions and Future Work

Learning patterns in data to predict branches seems to be a promising way to perform dynamic branch prediction. Although there has been extensive research in machine learning regarding improving accuracy on hard problems, not much attention has been drawn to the problem of real-time or anytime prediction. Modern computer architectures with their increasing clock speeds not only require branch predictors to work well but also require them to be of low latency. Perceptron predictors with Multiply-Add Contribution (MAC) seem to perform well under various workloads and in different scenarios. Also compared to *gshare* and *tournament* predictors the amount of hardware scales linearly with respect to the size of history.

In this report, we demonstrated the use of offline learning in workload and correlation detection by employing decision trees. This approach seems promising and a study of different information types in branch prediction needs to be explored more thoroughly. Whether the compiler could encode some information that would be helpful to the processor in dynamic branch prediction remains to be seen. Other techniques like using redundant history, making use of program counter information and use of multiple hash strategies via skewing have been suggested as a way of improving perceptron performance. Understanding these techniques in terms of machine learning algorithms may provide some insights into further improvements to these methods.

A general trend in learning based approaches to dynamic branch prediction (DBP) is inclusion of larger history sizes and more information types. *Offline* learning is an area which has been mostly unexplored in DBP. In applications where the stall caused due to branches may be critical

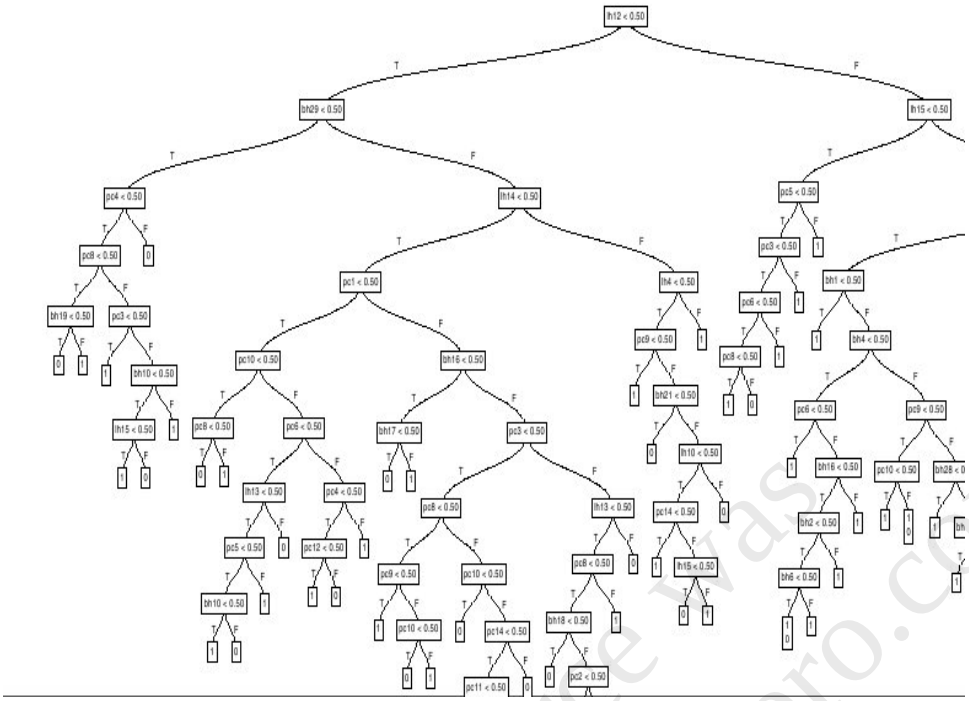

Figure 6: A part of a Decision Tree being learned.

the use of available traces could help tune the performance of the hardware branch predictor. The initial learning latency is also reduced by usage of offline traces available for these applications. Configuration of the predictor according to these traces may also be helpful. Other strategies for application of offline learning techniques apart from parameter tuning should also be explored.

5 Work Distribution

All the project tasks were divided evenly among the team members. The main division of labor was by algorithm:

- Kedar Bellare - Perceptrons, Kernel Perceptrons
- Pallika Kanani - Decision Trees
- Shiraj Sen - Local and Hybrid Predictors

The results and analysis of the algorithms and Adaptive Information Processing[3] was done together.

References

[1] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz and Y. Singer, "Online Passive-Aggressive Algorithms", *Journal of Machine Learning Research*, pp. 551-585, 2006.

[2] D. Jimenez and C. Lin, "Neural Methods for Dynamic Branch Prediction", *ACM Transactions on Computer Systems*, Vol 20., No. 4, pp. 369-397, Nov. 2002.

[3] H. Gao and H. Zhou, "Adaptive Information Processing: An Effective Way to Improve Perceptron Branch Predictors", *Journal of Instruction-Level Parallelism*, Vol 7. 2005.