Perceptrons Branch Predictor and some recent developments

Shugen Li
Department of Electrical and Computer Engineering, University of Alberta
Dec. 19th 2006

Abstract

In this paper, a new type of branch predictor is being introduced. It bases on the perceptrons, one of the simplest possible neural networks. The perceptrons uses the weights vector instead of the saturated counter at the traditional branch predictors to get the proper prediction and effectively increases the accuracy of the predictor. Also it can apply with longer branch histories without increasing the resources exponentially.

I will explain the basic principle of the perceptrons and simulate a single perceptrons predictor on the framework of the 2nd Championship Branch Predictor. Also it will be evaluated on the related traces. We will see that it improves the mis-predict rate by nearly 9.25% comparing with a single *Gshare* predictors at the same 64K hardware budget. And I introduce some current approaches about the perceptrons.

1. INTRODUCTION

As the new technology development on the modern processors, it trends to deeper pipeline and faster clock cycles to achieve higher performance. It means the necessary for more accurate branch prediction to avoid the following higher mis-prediction penalty and boost instruction-level parallelism on the modern computer architectures.

From figure 1 [1], we can see a general conceptual system model for the branch prediction that describes the dynamic branch prediction scheme. All source information, like branch address, local branch history, global history and other run time information, are collected during the program execution period. The information processor compresses a subset of these source information and gets an information vector. Thus, the predictor processes the information vector and makes a prediction. Traditionally, various Markov Finite State Machines (FSM) are used as the predictor [1], like correlated predictor, Gshare predictor, two-level predictor and others hybrid predictor. Most of them are using a table of two-bits (or more bits) saturate counters to make the related prediction and update it with the branch result. In the past decade, many works had been done to increase the prediction accuracy of this kind of predictors with a limited improvement following a huge and expensive hardware cost.

Apparently, a new way should be considered to get the significant improvement at branch prediction. Machine learning techniques offer the possibility of further improving performance by increasing prediction accuracy. We can improve accuracy by replacing these traditional predictors with neural networks, which provide good

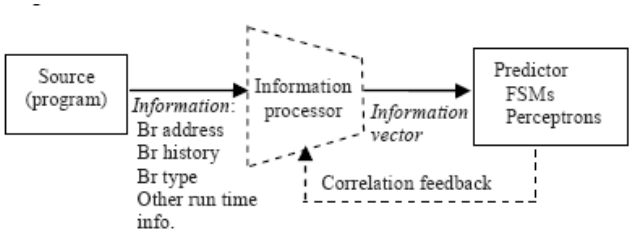


Figure 1. A conceptual system model for branch prediction [1].

predictive capabilities[2]. The first dynamic neural branch predictors (LVQ and MLP) were proposed by Lucian Vintan[3]. Daniel Jimenez and Calvin Lin consistenly research on the neural branch predictor and further developed the first perceptrons predictor that was feasible to implement in hardware. The perceptrons is one of the simplest possible neural networks -easy to understand, simple to implement, and have several attractive properties that differentiate them from more complex neural networks [2]. As we will see on the following sections, it applies a set of weights vector to replace the table of the two-bit saturate counter for getting more precise correlation between the branch history and prediction in order to achieve higher branch prediction accuracy. Also it can use longer branch history to obtain better performance only with the linearly increasing resources other than increasing the history length with exponential resources in two-bit counters methods.

The rest of the paper is organized as follows: Section 2 introduce the principle of the Perceptrons , its algorithm on the branch prediction and a limitation of the Perceptrons.. Section 3 will briefly discuss some methods how to implement this idea efficiently and describe a single Perceptrons predictor simulation experiment and discusses some behaviours from the results. Sections 4 will talk about some recent development on the Perceptrons branch prediction. Section 5 concludes the paper.

2. PERCEPTRONS BRANCH PREDICTOR

2.1 The attractive

The major reason to choose Perceptrons is that it can be effectively implemented on hardware besides others current neural networks. Also another advantage of perceptrons is that it is easy to understand the decisions that they make through evaluating their *weights*, i.e., the correlations that they learn. Many neural networks is difficult or impossible to determine exactly how the neural network is making its decision. Perceptron's

decision-making process is easy to understand as the result of a simple mathematical formula [2].

2.2 Perceptrons model and algorithm

A single-layer perceptrons , the simplest of many types of perceptrons , is used as our implement model . It is one artificial *neuron* connecting several input units by weighted edge to one output unit. A perceptron learns a target Boolean function t(Xo,…, X n) of n inputs[4]. Figure 2. show a graphical model of this perceptrons. Figure 3 express its algorithm.

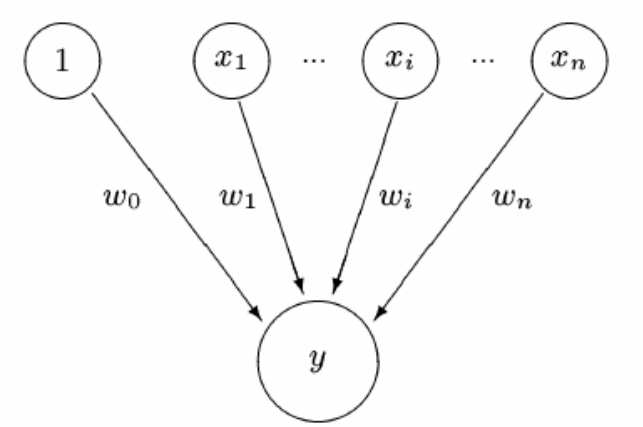


Figure 2: Perceptron model[4]

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

Figure 3: Perceptron algorithm[4]

In order to explain these ideas better, we define the below elements [2]:

- The input values (Xi,…,Xn) are the bits of the global branch history shift register(Xo default as 1, a bias always taken input). It can be either “1” as taken branch or “-1” as not taken.
- The weights vector (Wo,…,Wn) are the weights set reflecting the correlation between the branch being predicted and the branch results with the global history
- The output of the perceptrons (y) is the dot product of the weights vector and the input vector as showed on figure 3. if y>=0 means prediction is taken, otherwise it means not taken.

2.3 Perceptrons Training method

The below algorithm is used as for training the perceptrons as in figures 4. we can set $\mathbf{t}$ as “1” if the branch was taken, or “-1” if not taken. Θ is the threshold, a parameter to the training algorithm used to decide if it needs more training.

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```

Figure 4: Training algorithm[2]

When the branch result is consistent with Xi, the related weight Wi will increase; otherwise, Wi will decrease[2]. Through training, the weights vector will provide a more precise quantitative correlation between the global history and the current branch.

2.4 Perceptrons predictor block diagram

Figure 5 shows a block diagram for the perceptrons predictor. The processor keeps a table of N perceptrons in fast SRAM, similar to the table of two-bit counters in other branch prediction schemes. The number of perceptrons, N, is dictated by the hardware budget and number of weights, which itself is determined by the amount of branch history we keep. Special circuitry computes the value of y and performs the training [2]

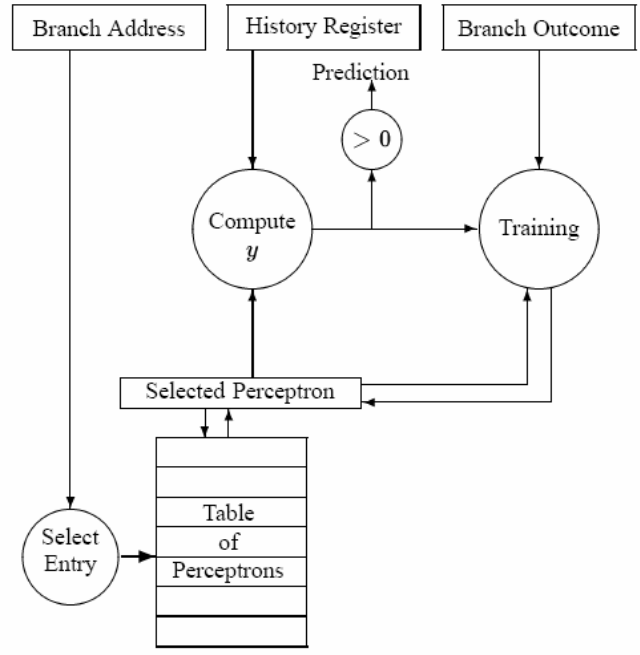


Figure 5: A block diagram[2]

The branch address is hashed to get an entry from the perceptrons table. The selected perceptrons, the related weights vector, will compute with global history register to produce their dot product , the output y. If y >=0, the prediction will be taken; otherwise not taken. After knowing the actual branch result, the related weights vector will be updated according to this outcome and the value of y through the training algorithm and put back on the table.

2.5 Perceptrons Training method

A limitation of perceptrons is that they are only capable

of learning *linearly separable* functions[5]. It means it can't perfectly reflect the non linearly separable branch behavior. For example, a perceptron can learn the logical AND of two inputs, but not the exclusive-OR.

Anyway most of branch can be described as linearly separable. Also some methods can be used to handle the linearly inseparable case as we will see the later developments. Overall it still shows better performance comparing with the traditional predictors.

3.IMPLEMENTATION AND EXPERIMENT

3.1 Effective implementation

We will see some effective ways on the actual implementation in the following paragraphs:

When computing the perceptrons, multiplication isn't need to compute the dot product as the input value Xi only is 1 or -1. We just need to add the weight on y when the Xi=1 and subtract it when Xi=-1[2]. This idea is easier to implement on hardware (adder circuits). Also, only the sign of the output is needed to make the prediction , so the others bit of the output can be performed a little slower.

The Training algorithm also can be simplified. First, all updating bit can be performed in parallel. Second, we can just increment Wi by 1 if t=Xi and decrement Wi by -1 if t≠ Xi as Xi and t can only be 1 or -1, This quick arithmetic operation also can be easily to implement[2].

3.2 Experiment setup

According to this idea, I wrote a simulation C++ code basing on the framework of the 2nd Championship Branch Prediction (my classmate Pang Gang helps me to correct some syntax errors) and run on its' related traces with varying parameters. The performance is evaluated by mis-predict rate(MPKI: mis-prediction per 1000 branch instructions. The weight is an 8 bit signed integer. Also I make a comparison with the Gshare predictor (provided by the 2nd CBP) on the same hardware budget to see the performance improvement and usage of longer history bits.

3.3 Analysis of some experiment results

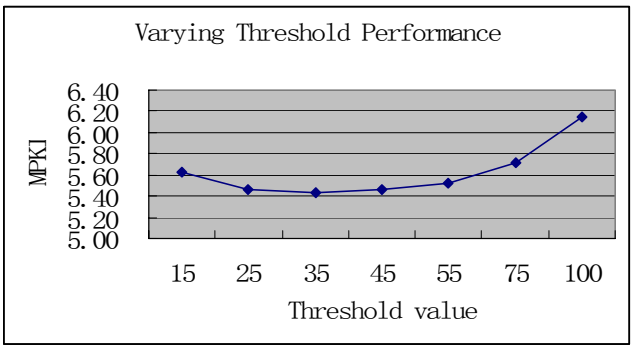
First, I keep the same history length and only increase the entry of table(hardware size). From the below table, we can see the performance improving with the bigger entry of table. It may means mis-predict rate improving by reducing aliasing as the No. of entries increase.

Hist-length	Entry	threshlod	Hwsize(K)	MKPI
16	256	45	32	5.818
16	512`	45	64	5.464
16	1024	45	128	5.295
16	2048	45	256	5.198
16	4096	45	512	5.156

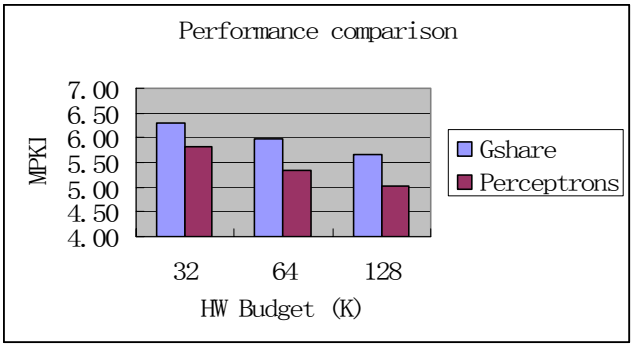
Second, I change the history length to observe the impact on performance in the following table. I found that if the history length is too long (in here as 60), it starts to lose the performance. There may be not so strong correlation with the longer history of global branch on these traces files and the weights generated cannot reflect the real connection. But we still see the significant improvement from 16 bits to 32 bits history length only with 2 times hardware budget.

Hist-length	Entry	threshlod	Hwsize(K)	MKPI
16	256	45	32	5.818
32	256	45	64	5.333
60	256	45	128	5.350

Next step is to find an appropriate threshold value θ to get the correct training period. The hardware is 64 K (16 bits history length, 512 Entries). We use a chart to find out where is located the best threshold value. According to this chart, we can see it is between 35 and 45, a little low than the best value (1.93h+14) from Jimenez. Further experiments are needed to get more accurate results.



At the end, I compared it with the Gshare predictor at the 32K, 64K and 128K Hardware budget. Gshare predictor only use 15, 16 and 17 bits length history and 16,32 and 32 Bits are being used on the perceptrons. Not only the better performance with the same Hardware cost the perceptrons predictor has, but also it can effectively use the longer global history. Overall the perception predictor have almost 10% improvement comparing with the Gshare.



4 .RECENT DEVELOPMENTS

However there still are some constrains as following for Perceptrons to implement practically:

- Like Delay-huge latency even if with simplified method
- low performance on the non linearly separable
- Trade off between Aliasing and high hardware cost.

Several people followed this basic idea to make further research on the perceptrons prediction. On the following

paragraphs, I will briefly introduce some new approaches to overcome these limitations.

4.1 Low-power Perceptrons (selective weight)

This paper is proposed by Kaveh Aasaraai, Amirali Baniasadi. They categorize the weights into three classes: Non-Effective, Semi-Effective and Highly-Effective. Mostly the prediction outcome would remain the same if only use the HE weights in the weight vector. So they suggest disabling NE and SE weights in the vector to reduce the overhead associated with perceptorns [6]. From the results they gave, some branch behave badly with this methods.
It maybe lacks of a little impractical.

4.2 The Combined Perceptron Branch Predictor

The author introduced a hybrid Perceptron predictor. It consists of two concurrent perceptron-like neural networks; one using as inputs branch history information, the other one using program counter bits. With the higher overhead, it achieved better performance than a single perceptrons predictor [7].

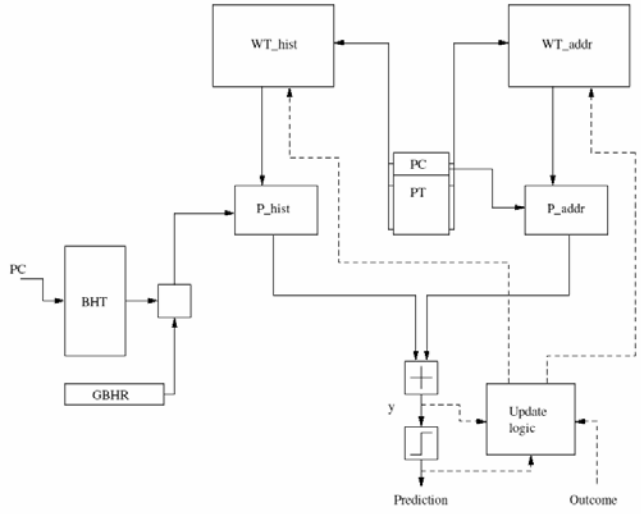


Figure 6: A block diagram [7]

4.3 Path-based neural prediction

Later Jimenez proposed this new approach with the following features [8].

- On an N-branch Path-Based Neural predictor, the prediction for a branch is initiated N-branch ahead. The predictions for the N next branches are computed in parallel.
- A row of N counters is read using the current instruction block address. On blocks featuring a branch, one of the read counters is added to each of the N partial sums.
- The delay is the perceptrons table read delay followed by a single multiply-add delay.

Anyway it doesn't consider the table read delay. The further development should be made to implement more practically and reduce the overhead.

4.4 Revisiting Perceptrons predictor

The below ideas were introduced by A. Seznec to deal with some perceptrons limitations[9]:

- using pseudo-tag to reduce aliasing impact
- skewing perceptrons weight tables to improve table utilization,
- introducing redundant history to handle linearly inseparable data sets.

Still the better performance is accompanying with the higher hardware complexity.

4.5 The O-GEometric History Length branch predictor

On the 1st CBP, A. Seznec proposed this new predictor and it was evaluated as the No.1 practical predictor. It use the perceptrons concept combined with some old techniques to obtain a better tradeoff between accuracy and the overhead.

- The GEHL predictor features M distinct predictor tables Ti
- The predictor tables store predictions as signed saturated counters.
- A single counter C(i) is read on each predictor table Ti.($1< i < M$)
- The prediction is computed as the sign of the sum S of the M counters C(i). like the first equation.
- The prediction is taken when S is positive or null and not-taken when S is negative
- The history lengths used the second equation for computing the indexing functions for tables Ti
- The element on all T(i) table is easy to train, similar like in the perceptrons predictor formula[10].

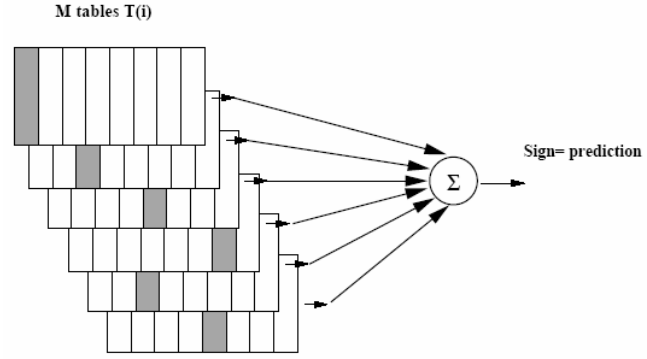


Figure 5: O-GEHL predictor[10]

The first equation is:

$$S = \frac{M}{2} + \sum_{0 \le i < M} C(i)$$ [1]

The second equation is:

$$L(i) = \alpha^{i-1} * L(1)$$

As we see, the predictor obtain the better performance and using of longer history with low hardware cost and better latency and make it to be a recent practical higher performance predictor.

5. CONCLUSION

Perceptrons is an easy and simple neural network. Its' implementation on the branch prediction had increased the predictor performance significantly by describing more

precise correlation between the history and the current branch. Also it is attractive as using long history lengths without requiring exponential resources. But its' weakness is the increased computational complexity and following latency and hardware cost.

There are several methods being developed to reduce the latency and the other constrains. As the new idea, it can be combined with the traditional methods to obtain better performance like the O-GEHL predictor. There should be more space for the further development. Anyway this technology will be more practical as the hardware cost go down quickly.

REFERENCE:

[1] I. K. Chen, J. T. Coffey, and T. N. Mudge, "Analysis of branch prediction via data compression", *Proc. of the 7th Int. Conf. on Arch. Support for Programming Languages and Operating Systems* (ASPLOS-VII), 1996.

[2] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons", *Proc. of the 7th Int. Symp. on High Perf.Comp. Arch* (HPCA-7), 2001.

[3] Branch predictor- Wikipedia, the free encyclopedia

[4] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962.

[5] L. Faucett. Fundamentals of Neural Networks: Architectures, Algorithms and Applications. Prentice-Hall, Englewood Cliffs, NJ, 1994.

[6] Kaveh Aasaraai, Amirali Baniasadi: Low-Power Perceptron Branch Predictor aasaraai@engr.uvic.ca , amirali@ece.uvic.ca

[7] M. Monchiero and G. Palermo The Combined Perceptron Branch Predictor Report n. 2004.35 Politecnico di Milano . Dipartimento di Elettronica e Informazione Via Ponzio, 34/5, 20133 Milan, Italy

[8] D. Jimenez , Fast path-based neural branch prediction. In *Proceedings of MICRO-36*, 2003.

[9] A. Seznec, "Revisiting the perceptron predictor",*Technical Report*, IRISA, 2004.

[10] A. Seznec. The **O-GEometric History Length branch predictor** In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004