# *Dynamic Branch Prediction with Perceptrons*

Nicolae-Andrei Vasile

*Faculty of Computer Science and Automatic Control*

*POLITEHNICA University of Bucharest*

Bucharest, Romania

***Abstract – This paper presents a new method for branch prediction. The key idea is to use one of the simplest possible neural networks, the perceptron, as an alternative to the commonly used two-bit counters. Our predictor achieves increased accuracy by making use of long branch histories, which are possible because the hardware resources for our method scale linearly with the history length. By contrast, other purely dynamic schemes require exponential resources.***

## I. INTRODUCTION

Modern computer architectures increasingly rely on speculation to boost instruction-level parallelism. For example, data that is likely to be read in the near future is speculatively prefetched, and predicted values are speculatively used before actual values are available. Accurate prediction mechanisms have been the driving force behind these techniques, so increasing the accuracy of predictors increases the performance benefit of speculation. Machine learning techniques offer the possibility of further improving performance by increasing prediction accuracy. In this paper, we show that one machine learning technique can be implemented in hardware to improve branch prediction.

Branch prediction is an essential part of modern microarchitectures. Rather than stall when a branch is encountered, a pipelined processor uses branch prediction to speculatively fetch and execute instructions along the predicted path. As pipelines deepen and the number of instructions issued per cycle increases, the penalty for a misprediction increases. Recent efforts to improve branch prediction focus primarily on eliminating aliasing in two-level adaptive predictors, which occurs when two unrelated branches destructively interfere by using the same prediction resources [1].

Our work builds on the observation that all existing two level techniques use tables of saturating counters. It's natural to ask whether we can improve accuracy by replacing these counters with neural networks, which provide good predictive capabilities. Since most neural networks would be prohibitively expensive to implement as branch predictors, we explore the use of perceptrons, one of the simplest possible neural networks. Perceptrons are easy to understand, simple to implement, and have several attractive properties that differentiate them from more complex neural networks.

## II. BRANCH PREDICTION WITH PERCEPTRONS

This section provides the background needed to understand our predictor. We describe perceptrons, explain how they can be used in branch prediction, and discuss their strengths and weaknesses. We then describe our basic prediction mechanism, which is essentially a two-level predictor that replaces the pattern history table with a table of perceptrons [2].

*The Perceptron Predictor*

The perceptron predictor uses a simple linear neuron known as a perceptron to perform branch direction prediction. The most accurate single-component branch predictors in the literature are neural branch predictors. Unfortunately, the high latency of the original perceptron predictor and makes it impractical for improving performance [3].

*Neural Branch Prediction Background*

Neural branch predictors keep a table of weights vectors, i.e., vectors of small integers that are learned through the perceptron learning rule [3]. As in global two-level adaptive branch prediction, a shift register records a global history of outcomes of conditional branches, recording true for taken, or false for not taken. To predict a branch outcome, a weights vector is selected by indexing the table with the branch address modulo the number of weights vectors. The dot product of the selected vector and the global history register is computed, where true in the history represents 1 and false represents -1. If the dot product is at least 0, then the branch is predicted taken, otherwise it is predicted not taken.

*How Perceptrons Work*

The perceptron was introduced in 1962 as a way to study brain function. The simplest of many types of perceptrons are considered, a single-layer perceptron consisting of one artificial neuron connecting several input units by weighted edges to one output unit. A perceptron learns a target Boolean function t($x_1$, …, $x_n$) of *n* inputs. In our case, the $x_i$ are the bits of a global branch history shift register, and the target function predicts whether a particular branch will be taken. Intuitively, a perceptron keeps track of positive and negative correlations between branch outcomes in the global history and the branch being predicted.

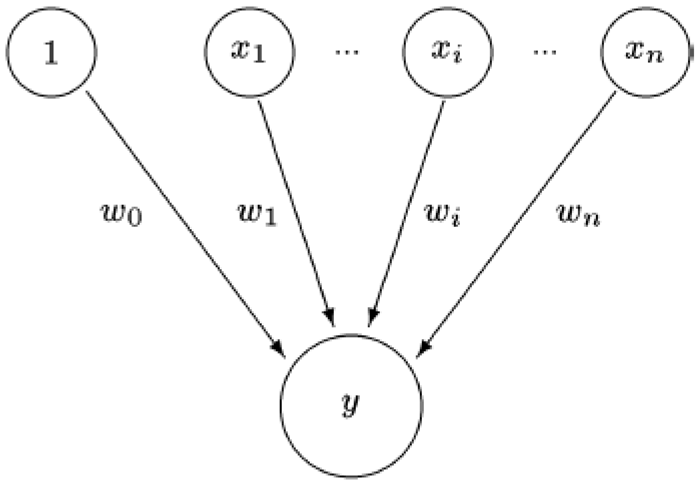

Figure 1. Perceptron model

Figure 2 shows a graphical model of a perceptron. The input values $x_1$, …, $x_n$, are propagated through the weighted connections by taking their respective products with the weights $w_1$, …,$w_n$. These products are summed, along with the bias weight $w_0$, to produce the output value *y*. A perceptron is represented by a vector whose elements are the weights. For our purposes, the weights are signed integers. The output is the dot product of the weights vector, $w_{0…n}$, and the input vector, $x_{1…n}$ ($x_0$ is always set to 1, providing a “bias” input).

The inputs to our perceptrons are bipolar; that is, each xi is either ¡1, meaning not taken or 1, meaning taken. A negative output is interpreted as predict not taken. A non-negative output is interpreted as predict taken.

*Using Perceptrons in Branch Predictors*

We can use a perceptron to learn correlations between particular branch outcomes in the global history and the behavior of the current branch. These correlations are represented by the weights. The larger the weight, the stronger the correlation, and the more that particular branch in the global history contributes to the prediction of the current branch. The input to the bias weight is always 1, so instead of learning a correlation with a previous branch outcome, the bias weight $w_0$ learns the bias of the branch, independent of the history.

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. When the processor encounters a branch in the fetch stage, the following steps are conceptually taken.

1. The branch address is hashed to produce an index *i* into the table of perceptrons.
2. The *i*th perceptron is fetched from the table into a vector register $P_{0\ldots n}$ of weights.
3. The value of *y* is computed as the dot product of P and the global history register.
4. The branch is predicted not taken when y is negative, or taken otherwise.
5. Once the actual outcome of the branch becomes known, the training algorithm uses this outcome and the value of y to update the weights in P.
6. P is written back to the *i*th entry in the table.

It may appear that prediction is slow because many computations and SRAM transactions take place in Steps 1 through 5. However, a number of arithmetic and microarchitectural tricks enable a prediction in a single cycle, even for long history lengths.

## III. ADVANTAGES OF THE PERCEPTRON PREDICTOR

We hypothesize that the main advantage of the perceptron predictor is its ability to make use of longer history lengths. Schemes such as *gshare* that use the history register as an index into a table require space exponential in the history length, whereas the perceptron predictor requires space linear in the history length.

To provide experimental support for our hypothesis, we simulate *gshare* and the perceptron predictor at a 64 KB hardware budget, where the perceptron predictor normally outperforms *gshare*. However, by only allowing the perceptron predictor to use as many history bits as *gshare* (18 bits), we find that *gshare* performs better, with a misprediction rate of 4.83% compared with 5.35% for the perceptron predictor [2]. The inferior performance of this crippled predictor is likely due to increased destructive aliasing, as perceptrons are larger, and thus fewer, than *gshare*'s two-bit counters.

Figure 9 shows the result of simulating gshare and the perceptron predictor with varying history lengths on the SPEC 2000 benchmarks. Here, we use a 4 MB hardware budget to allow gshare to consider longer history lengths than usual. As we increase history length, gshare becomes more accurate until it degrades slightly at 18 bits and then runs out of bits (since gshare requires resources exponential in the number of history bits). By contrast, the perceptron predictor's accuracy only improves with longer histories. With this unrealistically large hardware budget, gshare performs best with a history length of 17, where it achieves a misprediction rate of 3.4%. The perceptron predictor is best at a history length of 48, where it achieves a misprediction rate of 2.9% [2].

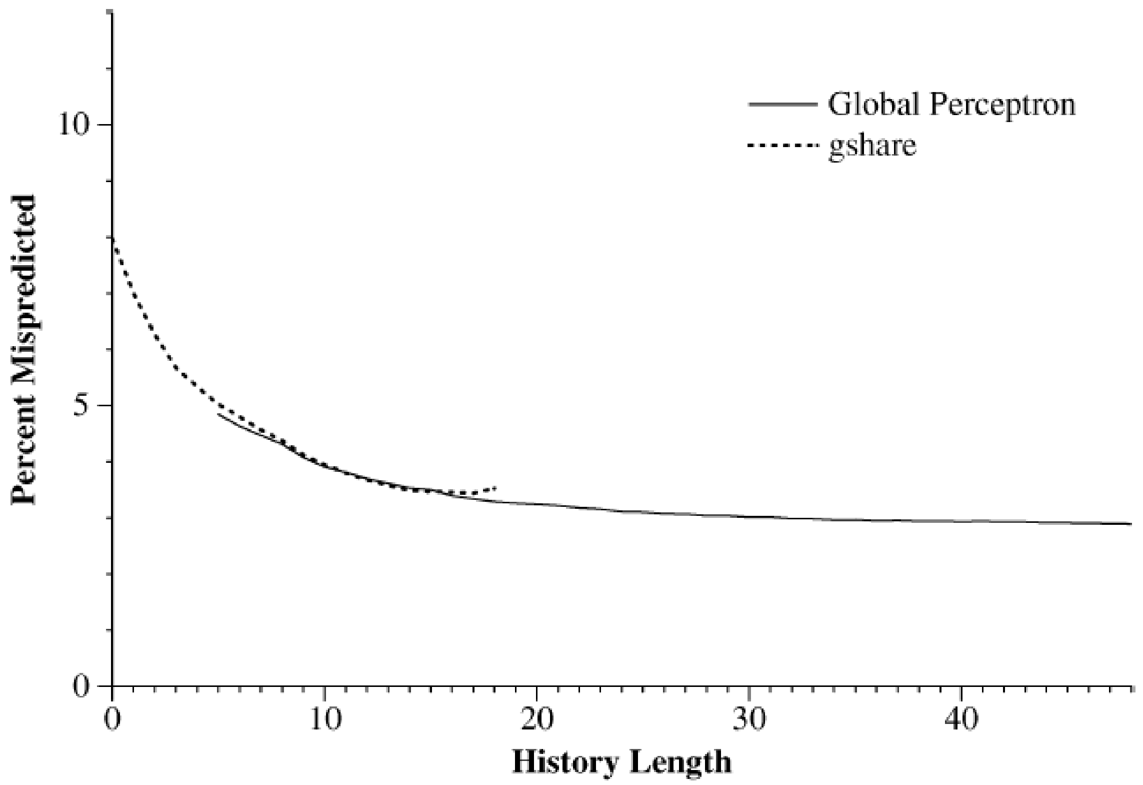

*Figure 2. History length vs. performance*

*Additional Advantages of the Perceptron Predictor*

1. **Assigning Confidence to Decisions.**
   Our predictor can provide a confidence level in its predictions that can be useful in guiding hardware speculation. The output y of the perceptron predictor is not a Boolean value, but a number that we interpret as taken if $y \geq 0$. The value of *y* provides important information about the branch since the distance of *y* from 0 is proportional to the certainty that the branch will be taken. This confidence can be used, for example, to allow a microarchitecture to speculatively execute both branch paths when confidence is low, and to execute only the predicted path when confidence is high. Some branch prediction schemes explicitly compute a confidence in their predictions, but in our predictor this information comes for free. We have observed experimentally that the probability that a branch will be taken can be accurately estimated as a linear function of the output of the perceptron predictor.
2. **Analyzing Branch Behavior with Perceptrons.**
   Perceptrons can be used to analyze correlations among branches. The perceptron predictor assigns a weight to each bit in the branch history. When a particular bit is strongly correlated with a particular branch outcome, the magnitude of the weight is higher than when there is less or no correlation. Thus, the perceptron predictor learns to recognize the bits in the history of a particular branch that are important for prediction, and it learns to ignore the unimportant bits. This property of the perceptron predictor can be used with profiling to provide feedback for other branch prediction schemes.

## IV. CONCLUSIONS

In this article we have introduced a new branch predictor that uses neural learning techniques—the perceptron in particular—as the basic prediction mechanism. The key advantage of perceptrons is their ability to use long history lengths without requiring exponential resources. These long history lengths lead to extremely high accuracy. A potential weakness of perceptrons is their increased computational complexity when compared with two-bit counters, but we have shown how a perceptron predictor can be implemented efficiently with respect to both area and delay. In particular, we believe that the most feasible implementation is the overriding perceptron predictor, which uses a simple Smith predictor to provide a quick prediction that can be later overridden. For an aggressive 1.76-GHz clock rate, this overriding predictor provides an IPC improvement of 5.7% over a McFarling-style hybrid predictor [2]. Another weakness of perceptrons is their inability to learn linearly inseparable functions, but we have shown that this is a limitation of existing branch predictors as well.

Finally, perceptrons have interesting characteristics that open up new avenues for future work. Perceptrons can also be used to guide speculation based on branch prediction confidence levels, and perceptron predictors can be used in recognizing important bits in the history of a particular branch.

## V. REFERENCES

[1] D. A. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, Monterrey, Mexico, 2001.

[2] D. A. Jimenez and C. Lin, "Neural methods for dynamic branch prediction," *ACM Transactions on Computer Systems*, vol. 20, no. 4, p. 369–397, 2002.

[3] D. A. Jimenez, "Piecewise Linear Branch Prediction," in *International Symposium on Computer Architecture*, Madison, WI, USA, 2005.