Dynamic Branch Prediction with Perceptrons

Daniel A. Jiménez Calvin Lin
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{djimenez,lin}@cs.utexas.edu

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.*

We describe our design and evaluate it with respect to two well known predictors. We show that for a 4K byte hardware budget our method improves misprediction rates for the SPEC 2000 benchmarks by 10.1% over the gshare *predictor. Our experiments also provide a better understanding of the situations in which traditional predictors do and do not perform well. Finally, we describe techniques that allow our complex predictor to operate in one cycle.*

1 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 [10, 24]. 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 [17, 16, 22, 4], which occurs when two unrelated branches destructively interfere by using the same prediction resources. We take a different approach—one that is largely orthogonal to previous work—by improving the accuracy of the prediction mechanism itself.

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.

We propose a two-level scheme that uses fast perceptrons instead of two-bit counters. Ideally, each static branch is allocated its own perceptron to predict its outcome. Traditional two-level adaptive schemes use a pattern history table (PHT) of two-bit saturating counters, indexed by a global history shift register that stores the outcomes of previous branches. This structure limits the length of the history register to the logarithm of the number of counters. Our scheme not only uses a more sophisticated prediction mechanism, but it can consider much longer histories than saturating counters.

This paper explains why and when our predictor performs well. We show that the neural network we have chosen works well for the class of *linearly separable branches*, a term we introduce. We also show that programs tend to have many linearly separable branches.

This paper makes the following contributions:

- We introduce the perceptron predictor, the first dynamic predictor to successfully use neural networks, and we show that it is more accurate than existing dynamic global branch predictors. For a 4K byte hardware budget, our predictor improves misprediction rates on the SPEC 2000 integer benchmarks by 10.1%.
- We explore the design space for two-level branch predictors based on perceptrons, empirically identifying good values for key parameters.
- We provide new insights into the behavior of branches, classifying them as either linearly separable or inseparable. We show that our predictor performs better on linearly separable branches, but worse on linearly inseparable branches. Thus, our predictor is complementary to existing predictors and works well as part of a hybrid predictor.

- We explain why perceptron-based predictors introduce interesting new ideas for future research.

2 Related Work

2.1 Neural networks

Artificial neural networks learn to compute a function using example inputs and outputs. Neural networks have been used for a variety of applications, including pattern recognition, classification [8], and image understanding [15, 12].

Static branch prediction with neural networks. Neural networks have been used to perform *static* branch prediction [3], where the likely direction of a branch is predicted at compile-time by supplying program features, such as control-flow and opcode information, as input to a trained neural network. This approach achieves an 80% correct prediction rate, compared to 75% for static heuristics [1, 3]. Static branch prediction performs worse than existing dynamic techniques, but is useful for performing static compiler optimizations.

Branch prediction and genetic algorithms. Neural networks are part of the field of machine learning, which also includes genetic algorithms. Emer and Gloy use genetic algorithms to "evolve" branch predictors [5], but it is important to note the difference between their work and ours. Their work uses evolution to design more accurate predictors, but the end result is something similar to a highly tuned traditional predictor. We propose putting intelligence in the microarchitecture, so the branch predictor can learn and adapt on-line. In fact, their approach cannot describe our new predictor.

2.2 Dynamic Branch Prediction

Dynamic branch prediction has a rich history in the literature. Recent research focuses on refining the two-level scheme of Yeh and Patt [26]. In this scheme, a pattern history table (PHT) of two-bit saturating counters is indexed by a combination of branch address and global or per-branch history. The high bit of the counter is taken as the prediction. Once the branch outcome is known, the counter is incremented if the branch is taken, and decremented otherwise. An important problem in two-level predictors is aliasing [20], and many of the recently proposed branch predictors seek to reduce the aliasing problem [17, 16, 22, 4] but do not change the basic prediction mechanism. Given a generous hardware budget, many of these two-level schemes perform about the same as one another [4].

Most two-level predictors cannot consider long history lengths, which becomes a problem when the distance between correlated branches is longer than the length of a global history shift register [7]. Even if a PHT scheme could somehow implement longer history lengths, it would not help because longer history lengths require longer training times for these methods [18].

Variable length path branch prediction [23] is one scheme for considering longer paths. It avoids the PHT capacity problem by computing a hash function of the addresses along the path to the branch. It uses a complex multi-pass profiling and compiler-feedback mechanism that is impractical for a real architecture, but it achieves good performance because of its ability to consider longer histories.

3 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. Our method is essentially a two-level predictor, replacing the pattern history table with a table of perceptrons.

3.1 Why perceptrons?

Perceptrons are a natural choice for branch prediction because they can be efficiently implemented in hardware. Other forms of neural networks, such as those trained by back-propagation, and other forms of machine learning, such as decision trees, are less attractive because of excessive implementation costs. For this work, we also considered other simple neural architectures, such as ADALINE [25] and Hebb learning [8], but we found that these were less effective than perceptrons (lower hardware efficiency for ADALINE, less accuracy for Hebb).

One benefit of perceptrons is that by examining their *weights*, i.e., the correlations that they learn, it is easy to understand the decisions that they make. By contrast, a criticism of many neural networks is that it is difficult or impossible to determine exactly how the neural network is making its decision. Techniques have been proposed to extract rules from neural networks [21], but these rules are not always accurate. Perceptrons do not suffer from this opaqueness; the perceptron's decision-making process is easy to understand as the result of a simple mathematical formula. We discuss this property in more detail in Section 5.7.

3.2 How Perceptrons Work

The perceptron was introduced in 1962 [19] as a way to study brain function. We consider the simplest of many types of perceptrons [2], 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 shows a graphical model of a perceptron. 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 output y of a perceptron is computed as

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