# Branch prediction using neural networks

Veerle Desmet and Koen De Bosschere

Ghent University
Department of Electronics and Information Systems (ELIS)
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium

E-mail: {vdesmet,kdb}@elis.rug.ac.be

## Abstract

*Modern processors use superscalar architectures with deep pipelines in order to execute multiple instructions per cycle. Feeding these pipelines with a constant flow of useful instructions is crucial for the overall processor performance. Branch instructions interrupt this feeding process as their outcome determines the next instruction to be executed. Modern microarchitectures overcome this difficulty by using a branch predictor, which predicts the direction of a branch in the fetch unit. Many prediction schemes have been proposed to predict accurately the behaviour of branch directions. Recently Jiménez* et al. *introduce the perceptron predictor [1], the first dynamic predictor to successfully use neural networks. Moreover, this simple neural network achieves higher accuracies compared to other predictors. In current research we explore some alternative design possibilities for branch predictors using neural networks.*

## 1. Introduction

*Deeply pipelined computer architectures as we know them today rely on a fetching mechanism that provides one or more useful instructions every clock cycle. Conditional branch instructions cause difficulties in this constant feeding process: the next instruction to be executed is not known until the branch condition (e.g. argument equals zero) is computed. This computation typically completes 3-14 cycles after the branch has entered the pipeline, meanwhile no further instructions can be fetched.*

*To solve this situation, an essential part of modern microarchitectures consists of branch prediction. A branch predictor predicts the outcome of the branch condition so that instructions on the predicted path can enter the pipeline the next cycle. This method enables a constant pressure on the pipeline but involves additional complications for handling speculative instructions and verifying the prediction. Of course the branch instruction itself is executed to verify the prediction. On a correct prediction all speculative instructions are useful and finish earlier compared to no branch prediction. On a misprediction however, all speculative work has to be undone before the correct path executes. This means that on a misprediction there is even an extra penalty compared to no branch predictor. As pipelines deepen and the number of instructions issued per cycle increases, the penalty for a misprediction also increases. Current branch predictors already reach 95% prediction accuracy and continuously improvements and new directions are exploited in this domain. The driving force behind these efforts is the large improvement in the average number of instructions executed per cycle (IPC) that is possible by even a small improvement in prediction accuracy [4].*

## 2. Related work

*During the last 10 years many research in the domain of branch prediction was done to improve branch prediction accuracy. Important contributions include the proposal of the* gshare predictor *by McFarling in [3] and the design*

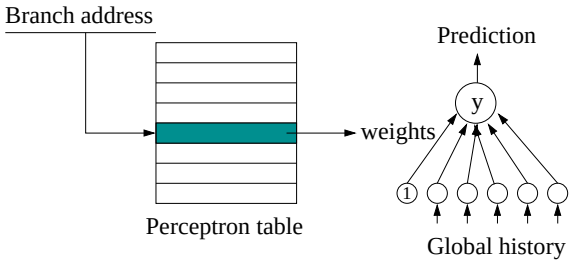


Figure 1: Perceptron predictor

*of* two-level predictors *by Yeh and Patt [5]. Gshare makes a prediction based on the XOR-combination of branch address and global history and is considered as one of the best predictors known today (except for hybrid predictors). Two-level predictors are important because from their organization follows that some latency in a branch predictor can be tolerated and nevertheless deliver a prediction in a single cycle. Other work mainly consists of refinements on these prediction techniques.*

## 3. Perceptron predictor

*In 2001, Jiménez* et al. *introduced a perceptron branch predictor [1] that uses a single layer perceptron as shown in Figure 1. The prediction is based on global history information, i.e. the outcomes of previously resolved branches. Each node in the input layer of the perceptron represents one bit of the global history,* $1/-1$ *if the corresponding branch was taken/not-taken. One input is always set to 1, providing a bias input. The vector of weights is taken from the perceptron table indexed by the branch address. The perceptron output y is computed as the dot product of weights and input vector and serves to decide whether the branch is predicted taken* ($y \geq 0$) *or not-taken* ($y < 0$).

*During update a weight is incremented when the branch outcome agrees with the corresponding input bit, and is decremented when it disagrees. The stronger the correlation, i.e. agreement or disagreement, the larger the weight and the higher its contribution to the output and final prediction. The weights are only updated on a misprediction or when y holds a small value, which indicates a poorly convinced prediction. A single layer perceptron is easy to understand but deals with the limitation that only so-called linearly separable functions can be perfectly learned.*

## 4. Own contribution

*In current research we explore some key aspects in the perceptron predictor and compare them with alternative options.*

### 4.1. Learning

*The learning aspect is certainly the main point in a neural network. Designing the learning part asks precise definitions to: when to learn and how to learn.*

*The learning algorithm can vary from easy incrementing/decrementing weights like proposed in the perceptron predictor to more complex adaptation of weights in order to fulfill certain error criteria. As mentioned in [1], the learning algorithm must be efficiently implementable in a branch predictor. Therefore simply incrementing/decrementing is preferable as the adaptation of all weights can be done in parallel.*

*The second point in learning is the condition on which further learning is forced. The condition is strongly recommended to avoid overtraining; possible conditions we already examined include:*

- *Only learn after a misprediction, a.k.a. error correcting procedure*
- *Only learn after a misprediction or when the output value does not reach a certain threshold* $\Theta$*, i.e. this strategy that is used in the perceptron predictor*
- *Keep on training*

*We evaluate different learning conditions while using the update strategy proposed in the perceptron predictor. During these experiments we measured behind prediction accuracy also the time spent in perceptron training. While the third learning condition causes a 100% training time (learn after each prediction), the other strategies spent less than 10% in training and obtain better prediction accuracies. We also studied the influence of history length and found that the time spent in training decreases when longer global history lengths are used. In all cases, the use of longer history length achieves higher accuracies although the size of the predictor increases: the length of the global history determines the number of weights that has to be stored in each entry of the perceptron table.*

### 4.2. Representation of weights

*During the design phase one should determine the representation of the weights: in case of the perceptron predictor signed integers are used limited to 10 bits. We measured the influence on both prediction accuracy and time spent in learning in case we use less bits to represent the perceptron weights.*

*Our experiments show that the accuracy not significantly decreases when e.g. 7 bits are used. Error correcting learning is appropriate when further reduction in bits is needed for the representation of weights. For the keep-on-training strategy we found a result of the overtraining property: at a certain point the prediction accuracy decreases when more bits are available.*

### 4.3. Confidence

*Neural networks provide an additional advantage: a confidence-level for free. Indeed, the network output is not a Boolean value but a number proportional to the certainty that the branch is taken. Some classical branch predictors explicitly add confidence for energy reduction as pointed out in [2]. We studied the quality of this own confidence technique and compared it with other confidence mechanisms like saturating counters. For a wide range of confidence thresholds the own confidence technique provides better results over adding traditional confidence.*

## 5. Acknowledgements

*Veerle Desmet is supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT).*

## References

*[1] Daniel A. Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In* Proceedings of the 7th International Symposium on High Performance Computer Architecture, *pages 197–206, January 2001.*

*[2] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline gating: Speculation control for energy reduction. In* Proceedings of the 25th Annual International Symposium on Computer Architecture, *pages 132–141, 1998.*

*[3] Scott McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, June 1993.*

*[4] Henk Neefs, Koen De Bosschere, and Jan Van Campenhout. An analytical model for performance estimation of modern data-flow style scheduling microprocessors. In* Proceedings of the 22nd Euromicro Conference: Short Contributions, *pages 2–7, 1996.*

*[5] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive training branch prediction. In* Proceedings of the 24th Annual International Symposium on Microarchitecture, *pages 51–61, November 1991.*