Machine Learning of Dynamic Branch Prediction

Yefan Tao, Ziyu Shu

Dec 1, 2019

1 Abstract

Our main goal of this project is to apply Machine Learning techniques(LSTM) on the problem of branch prediction, and compare the results with local predictor, Gshare predictor and perceptron predictor.

2 Introduction

Modern computer architectures increasingly use speculation methods to boost instruction-level parallelism. Data and values are likely to be used in the near future is speculatively prefetched or stored [1], [2]. Accurate prediction mechanisms have been the driving force behind these techniques, so increasing the accuracy of predictors increases the performance benefit of speculation. We have learnt local predictor, correlated predictor,tournament predictor, G share predictor and tagged hybrid predictor. These traditional techniques effectively increase the accuracy of the prediction, thus boost the performance of the whole system.

Due to its excellent performance in many other areas, machine learning techniques offer the possibility of further improving the accuracy of prediction. In this paper, we are going to compare some exist machine learning based prediction method with the traditional methods. Then, we are trying to apply some state-of-the-art machine learning algorithm to see if we can go a step further.

3 Research Plan and Methodology

We will use the input data from Daniel's work [3], which is available from GitHub [4]. Results on all different algorithms will be tested on the SPEC 2000 integer benchmarks. [5]

3.1 correlated predictor and local predictor

we will to implement a (2,2) correlated predictor and a (0,2) local predictor. Apply our test sample on it, and evaluate their performance: including space, speed, and correctness. This should be trivial.

3.2 gshare predictor

In this part, a gshare predictor will be implemented. Figure 1 shows the schematic structrue of a gshare predictor. Gshare predictor is one of the best purely dynamic global predictors from the branch prediction literature. Unlike other correclated predictor or local predictor, the gshare predictor will form the index by taking XOR of branch address and most recent branch outcome. So it make use of information both from the history and branch address. Ideally, G share predictor should provide a improvement on the previous methods.

3.3 dynamic branch prediction with perceptrons

we will implement some machine learning based predictor mentioned in Daniel's work [3]. Figure 2 shows the schematic structrue of a percetron. The benefit of a perceptron-based predictor is that, it's easy to implement and intepret the outcome. Compared to other methods like decision tree, perceptron is can be implemented efficiently, each node in the perceptron scheme can be a previous branch outcome. Also, the outcome of perceptrons is easy to track, the outcome is simply a overall summation on preivous branch result with a certain weight. It is claimed that the proposed algorithm provides an improvement of 10% over the G share predictor, and we will try to recover this improvement.

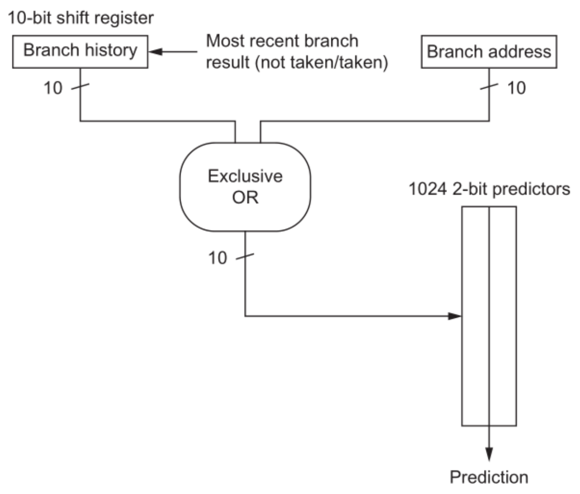


Figure 3.4 **A gshare predictor with 1024 entries, each being a standard 2-bit predictor.**

Figure 1: Schematic structure of a 10-bit gshare predictor(from textbook)

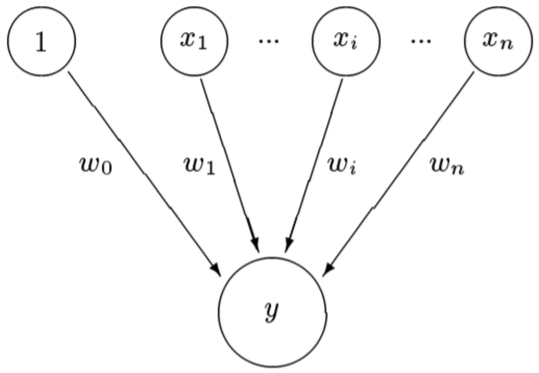
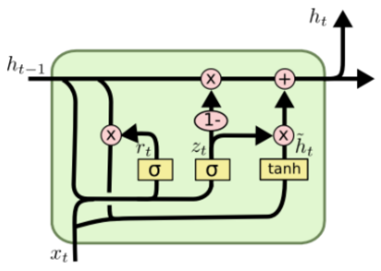


Figure 1: Perceptron Model. The input values $x_1, \ldots, x_n$, are propagated through the weighted connections by taking their respective products with the weights $w_1, \ldots, w_n$. These products are summed, along with the bias weight w_0, to produce the output value y.

Figure 2: Schematic structure of a perceptron predictor



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: Schematic structure of LSTM

3.4 advanced machine learning based predictor

Lastly, we try to implement some state-of-the-art machine learning algorithm , particularly, Long short-term memory (LSTM) [6] to see if we can further improve the performance. Figure 3 gives a typcial structure of how LSTM looks like: essentially it's a state machine consists of three gates. An input gate controls how many branch history the predictor is going to take into account, the output gate controls how many branch history will be actually used to calculate the prediction, and the forget gate decides whether we need to throw away some of the branch history. After several epoches of training, the LSTM should be able to make better prediction than other methods.

4 Results

Our script and test samples are available on **GitHub** [7]. The datasets we are using is from SPEC2000 [5]: 'GCC' and 'MCF' are two sample datasets taken from real CPU running. To make results comparable, we give different predictors the same parameters. However, different predictors have different setting for parameters, so we will explain the meaning of them in detail. In general, we give two parameters('l' and 'mode') to the predictors:
local predictor: local predictor only uses the parameter 'mod', which represent the number of branches. For each branch, we are using a 2-bit predictor(with 4 states).
gshare predictor: gshare predictor only uses the parameter 'l', which represent the length of bits used to do the XOR calculation. For example, if l=4, our XOR operation will be performed over a 4-bit binary number, resulting

in 32 possible branch address, and each of them is a 2-bit predictor.
perceptron predictor: perceptron predictor uses both 'l' and 'mod', the usage of 'mod' is the same as local predictor, meaning the number of branches. The parameter 'l' means the length of branch history used to update the perceptron.
LSTM predictor: LSTM predictor also uses both 'l' and 'mod', 'l' is the length of branch history and 'mod' is the number of branches.

Figure 4 and Figure 5 show the comparison between different predictor on 'GCC' and 'MCF', respectively, with l=2.

The first thing we can see is that, there're only three curves, that's because 'local' is highly overlapping with 'gshare'. For the same settings, the only difference between 'local' and 'gshare' is how to choose which branch address we should go into. Our results shows that doing a 'XOR' operation doesn't actually improve the accuracy, which is not expected. It's possible that our test sample are not large enough to demonstrate the superiority of 'gshare predictor'.

Secondly, under all conditions 'perceptron' is better than 'local' and 'gshare'. This is understandable, since the 'perceptron' make use of branch history in a 'smarter' way.

Thirdly, our 'LSTM' outperforms other predictors with low number of branches. When it goes to high number of branches, it seems sample-dependent: 'LSTM' is the worst for 'GCC' but the best for 'MCF'. This indicates the 'LSTM' is not stable. At least, for the training and testing sample we are using, we haven't found a stable parameter setting to let the 'LSTM' give a stable improvement over other predictors.

From another perspective, we set the number of branches to be the same('mod'=2) and let the parameter 'l' vary, what we get are shown in Figure 6 and Figure 7. We get what we expect on 'GCC' dataset, simply in terms of accuray, $'LSTM' >' perceptron' >' gshare' \approx' local'$, and on 'MCF' dataset, we have $'LSTM' \approx' perceptron' >' gshare' \approx' local'$.

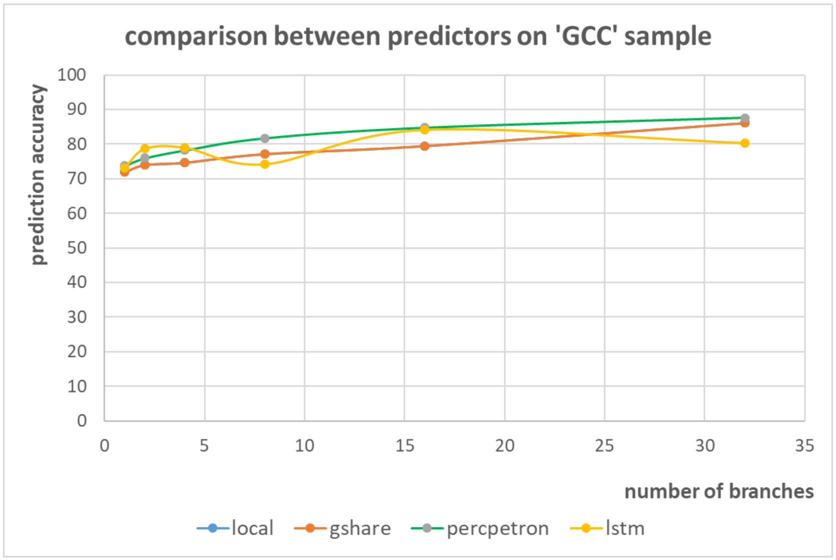


Figure 4: Comparison between different results on ’GCC’ with the same ’l’

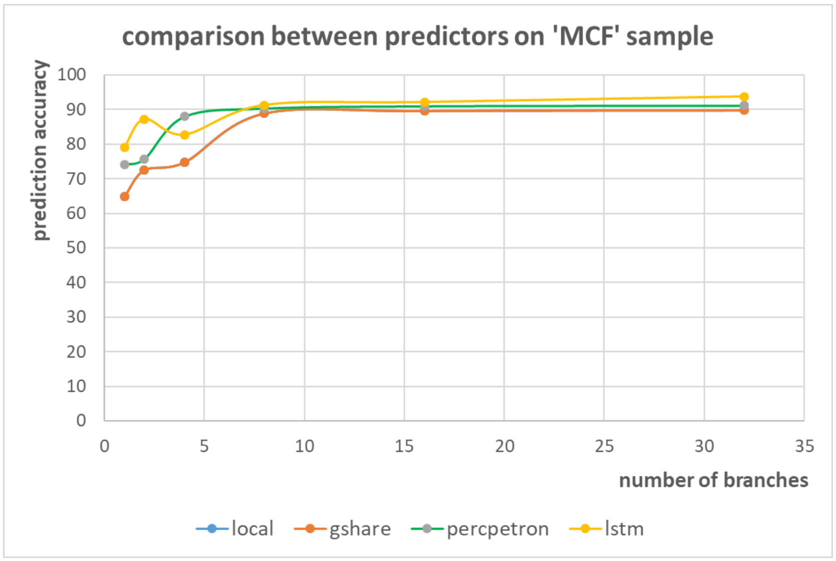


Figure 5: Comparison between different results on ’MCF’ with the same ’l’

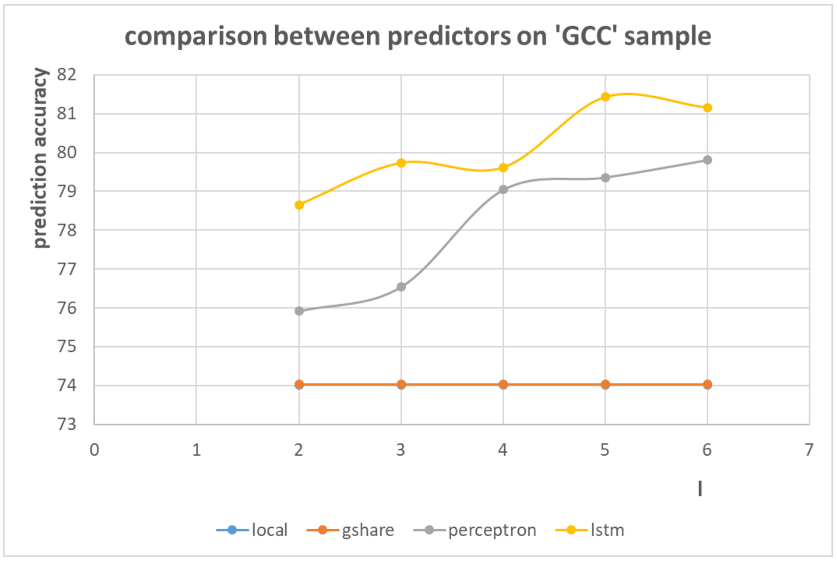


Figure 6: Comparison between different results on 'GCC' with the same 'mod'

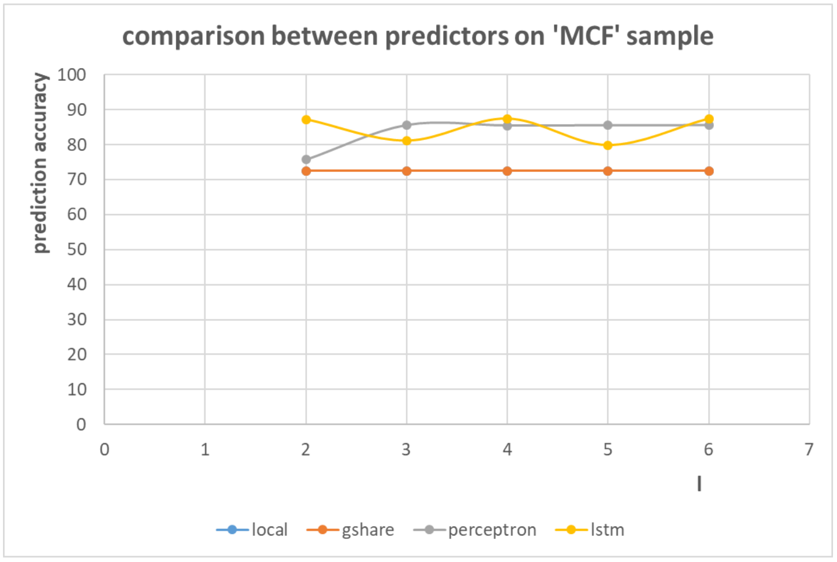


Figure 7: Comparison between different results on 'MCF' with the same 'mod'

5 Conclusion

In this report, we discuss about four different branch predictors and analyze their properties. To compare their actual performance, we implement them in Python and run test over SPEC2000 dataset. Though our results are not 100% as what we expect, we learn a lot from the experience and have better understanding how branch prediction is working in real CPU running.

6 Literature Survey

Below are the summary of reference mentioned above:(Given by the order in **References**)

1. [1] is our text book. From which we learnt a lot of basic traditional predictor such as (m,n)-correlated predictor, local predictor, G share predictor.

2. [2] is a instruction about a specific microprocessor.

3. [3] uses the basic perceptron to build a neural network. The author claims that it provides a 10% improvement compared to the traditional predictor.

4. [4] is the github address for our test sample.

5. [5] is the official website of SPEC 2000, which will be used for referrence for performance test.

6. [8] is used for learning the property and advantange of different kinds of machine learning algorithm.

7. [6]. We suppose that the branch prediction may also has some kind of long term and short term property, thus LSTM algorithm may fit it very well.

References

[1] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[2] R. E. Kessler, E. J. McLellan, and D. A. Webb. The alpha 21264 microprocessor architecture. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, pages 90–95, Oct 1998.

[3] D. A. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, Jan 2001.

[4] https://github.com/edougal/nn_bp_test.

[5] spec. https://www.spec.org/cpu2000/cint2000/.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[7] https://github.com/yefantao/branch_predicton.

[8] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.