# Neural Networks Branch Prediction with Limited Hardware Budget

**Jaeseok Huh**
jaeseok.h@kaist.ac.kr

*School of Computing, KAIST*

## Abstract

The recent hot trend in deep neural networks brings researchers to think about harnessing its effective prediction power across literature. In computer architecture, prediction and speculation arise as natural subjects of interest and have played an instrumental role in maximizing data level parallelism alongside pipelining scheme. An environment of hardware world is, however, totally different from that of software's application layer thereby calling forth a distinctive challenge: how to adopt it "efficiently" to cope with limited hardware budget? In this paper, we explore various neural-network-based methods for branch prediction task with the Championship Branch Prediction (CBP) 2016 workload and an in-house simulator.

## 1. Introduction

Recently, there have been significant breakthroughs in deep neural networks over computer vision [1], natural language processing [2], and speech recognition [3]. Its application has been fuelled by the data that have been stacked over decades in a wide array of areas in our society, from autonomous driving [4], recommendation system [5], medical image analysis [6], mobile advertising [7], fraud detection [8], fake news [9], and social sciences [10] , to military robots [11].
Retrieving appropriate dataset, labeling, annotating, and preprocessing in such areas still cost a significant amount of time and money, and may face ethical challenges related to privacy and informedness of the party concerned [12]. For computer architecture problems, however, it is much easier to obtain dataset and evaluate model. There is not much hassle of data processing due to its nature. So, why don't we try to adopt it? The problem here is the scarcity of silicon budget knowing that the size of cache memory is a determinant factor of modern architecture. On the other hand, the architecture community adopted pipelining that introduces control hazard (and other hazards) but also put an effort to minimize its deteriorating effect to overall CPI. The problem that multiple instructions are overlapped results in the uncertainty of which instruction to be executed right after the issuance of branching instruction. Taking into account that dynamic branch accounts for 15-25% and that it takes much longer time to resolve a branch that involves floating-point unit than integer unit, branch prediction would vastly contribute to reducing overall CPI. As a result, static compiler-based methods have been proposed, such as loop unrolling and instruction reordering. As not all of the hazard cannot be eliminated in such

ways, predictive methods including static prediction at compile-time and delayed branch are also developed. In those ways, a portion of CPU cycles that would be wasted otherwise can be saved with a time constraint that the prediction should be done with in a few microcycles (one or two usually).
In this paper we try to explore solutions to deal with i) the silicon storage budget and ii) limit of allowed time, focusing on branch prediction among a number of prediction and speculation problems in modern computer architecture.

## 2. Related Work

Machine learning for branch prediction using “Learning Vector Quantization algorithm” and multi-layer perceptron, called “neural branch prediction”, was first proposed by [13]. It was followed by [14] and further developed by [15]. The main advantage of the neural predictor is its ability to exploit long histories while curbing resource growth in linear cost using hash table, whereas classical predictors require exponential resource growth. In another work [16], by combining path and pattern history to overcome, it reports a global improvement of 5.7% over a McFarling-style hybrid predictor [17] with a gshare/perceptron overriding hybrid predictors. Deviating from (deep) feedforward networks widely being used in the aforementioned applications and branch predictions, recurrent neural network(RNN) is also of interest because of its ability to make use of their internal state (memory) to process sequences of inputs and retain them over time. As traditional approaches to branch prediction [19, 20] indicates the importance of tracking and correlating branch history, RNN provides the possibility of improvement. However, training such recurrent networks is not straightforward, since now the output of the network (and therefore the gradient) depends on the input, *and* all the previous inputs to the network through time scale. One approach used in the machine learning literature is to “unroll” the network through time and use conventional backporgation (see Figure 1.) This method is called and backpropagation through time (BPTT). To avoid unrolling to the beginning, one can instead go back through *h* steps ago. But this is inapplicable to our problem in which prediction must be done within a few (optimally one or two) cpu cycles and locality of communication is of another consideration. In order to resolve this disadvantage, real time recurrent learning (RTRL) is proposed to store the derivatives of only the previous interval as like Equation 1. [21]

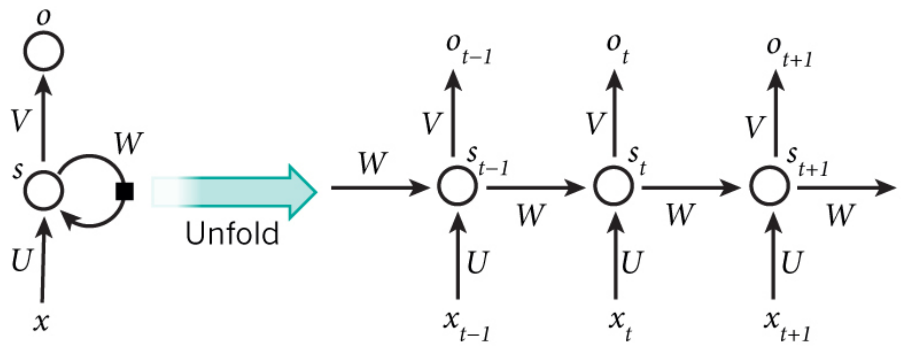

**Figure 1**. RNN Unfolding

$$\frac{\partial u_k^{(T+1)}}{\partial w_{ij}} = f'(s_k^{(T)}) \left[ \sum_{v \in U} w_{kv} \frac{\partial u_v^{(T)}}{\partial w_{ij}} + \delta_{ik} z_j^{(T)} \right]$$

**Equation 1**. Real Time Recurrent Learning (RTRL)

## 3. Approach

In this work, we explore three methods: (i) perceptron (single layer without hidden unit), (ii) 2-layered neural network, and (iii) TAGE-SC-L [25].

### 3.1. Perceptron

We replicate [15] with various hyperparameter and 8KB/32KB hardware budget. In Figure 2, there are *d* inputs ($I_1, I_2, \ldots I_d$) in addition to one bias unit (denoted *1*). The value of them is either 1 (taken) and 0 (not taken) and updated in a circular fashion. The weights (denoted $w_1, w_2, \ldots, w_d$) are initialized in normal distribution, with the standard deviation being the square root of the difference between maximum and minimum value of a unit. ($\sigma=2^4$ for 8-bit integer representation) The weighted sum is then fed into a unit. If the value is positive, it means the branch is predicted taken and not taken otherwise. After the actual result arrives, the weights are updated by backpropagation algorithm [26]. As the gradient updates are all independent, it can be implemented parallelly in hardware consuming only one or less microcycle.

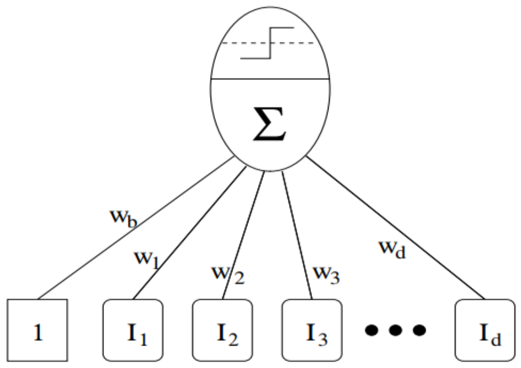

**Figure 2**. Perceptron Branch Predictor (based on [15])

The aforementioned elements construct one perceptron, which becomes an element of a hash table. The key of the hash table is a branch address. We use a hash function that can be calculated within a fraction of microcycle (shifting operation). Each unit of the perceptron relates to the possibly correlating branch so that the prediction can be performed based upon the *d* units. Because the storage budget is limited, we note that we can only use the limited size of input and hash table. The input size should be long enough to represent relevant branch history and hash table should be large enough to prevent frequent hash collision. We study the effective size of input and hash table through extensive experiment.

### 3.2. 2-layered Network

Perceptron is simple and light but not capable of learning linearly inseparable patterns, e.g. XOR. Hence, we also adopt 2-layered network to test its predictive power. The principle is the same as perceptron but we are limited to use a fewer number of units and weights. Although we do not perform simulation in circuit-level but conjecture that the prediction will take approximately twice as the time for perceptron and that so will the backpropagation.

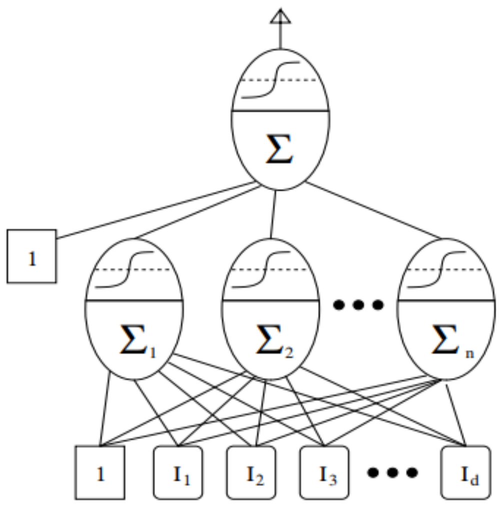

**Figure 3**. 2-layered Network

### 3.3. TAGE-SC-L [25]

Recent branch predictors incorporate more prediction information (i.e. path history and the like) and sought more efficient data structure, such as an adder tree [22, 23]. The predictor of TAGE [24] features tagged predictor components indexed with distinct global/local history lengths forming a geometric series. In the following work, TAGE-SC-L [25] further associates with small adjunct predictors by a statistical corrector (SC for short) and/or a loop predictor (L for short). However, it should be noted that this method requires longer microcycle or more stalls due to its multi-stage architecture and the larger number of the predictor that should be marshalled.

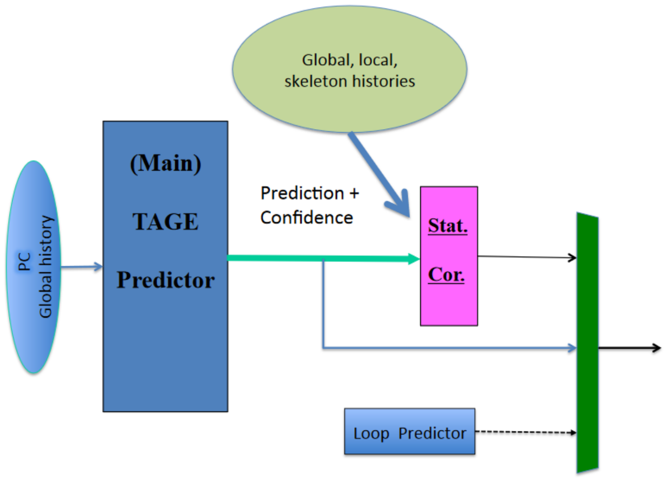

**Figure 4**. TAGE-SC-L

## 4. Experiments

### 4. 1. Environment

We set up an environment as follows:

```
Ubuntu 18. 04. 2 LTS
Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz
Memory 32GB
g++ 7.4.0
```

Also, we varied the storage budget to 8KB, 32KB, and indefinitely. Due to time limit, we tested some of the possible parameter combination rather than doing all of them. We kept CPU and memory as idle as possible throughout our experiment.

### 4. 2. Workload and Simulator

We use the Championship Branch Prediction(CBP) 2016 workload and modify its simulator. The workload is actually split into train and test set but we use both as it is online learning task. It is comprised of instructions for both mobile and server. Short ones consist of about 100 million instructions and long ones about 1 billion instructions. In total, the workload includes nearly 80 billions of instruction. The number of each type is shown in Table 1.

| Workload Size | Workload Type | # |
|---|---|---|
| Short (400) | Mobile | 293 |
| | Server | 107 |
| Long (40) | Mobile | 32 |
| | Server | 8 |

**Table 1.** The number of each workload type

## 5. Result

We first try to find the appropriate bits of representation with perceptron and a partial set of workloads. While inspecting the value of units and the final misprediction rate, we conclude that 8 bits of unsigned integer can effectively represent the neuron. The following results are all tested with 8-bit unsigned integer representation.

### 5. 1. Perceptron

| # of input | 8KB (# of perceptrons) | 32KB (# of perceptrons) |
|---|---|---|
| *Predict all taken* | 8.383% (-) | |
| 8 | 0.7332% (1,024) | 0.7123% (4,096) |
| 16 | 0.6536% (512) | 0.6226% (2,048) |
| 32 | **0.6226% (256)** | 0.5735% (1,024) |
| 64 | 0.6419% (128) | **0.5631% (512)** |
| 128 | 0.7084% (64) | 0.5850% (256) |

Table 2. Misprediction Rate with Percpetron

The results indicate that the imposed budget limits the accuracy of prediction. Given [15], which reports that the prediction accuracy does not improve significantly when the size of input gets larger than around 62, it is suggested that the misprediction rate can be lowered if we increase the number of perceptron (or the size of hash table).

**5. 2. 2-Layered Network**

| # of hidden units | 8KB / (# of input) = 30 | 32KB / (# of input) = 62 |
|---|---|---|
| *Predict all taken* | 8.383% (-) | |
| 1 | **2.7432%** | **4.0724%** |
| 2 | 2.9987% | 4.2473% |
| 4 | 4.3333% | 4.2945% |
| 8 | 4.3070% | 4.3518% |

Table 3. Misprediction with 2-layered Network

The existence of hidden unit seems to actually increase the misprediction rate. Tracking the value of the units, we found that the hidden units get saturated and the sign thereof hardly changes. Thus, we increased the representation bits to 16, 24, and 32 bit but failed to observe significant improvement. We conjecture that 2- (or possibly more) layered network is too complex model to predict branching result thereby causing overfitting and that linearly inseparable patterns such as XOR are not that frequent.

**5. 3. TAGE-SC-L**

We report only unlimited storage experiment: the misprediction rate was 0.2284%. This is close to the results reported in the original paper [25], 0.1782%, with different workloads. We did not test with 8KB/32KB storage constraint due to our limited time. The original paper reports 0.3315% with 32KB of storage constraint.

## 6. Discussion

### 6. 1. Why not RNN?

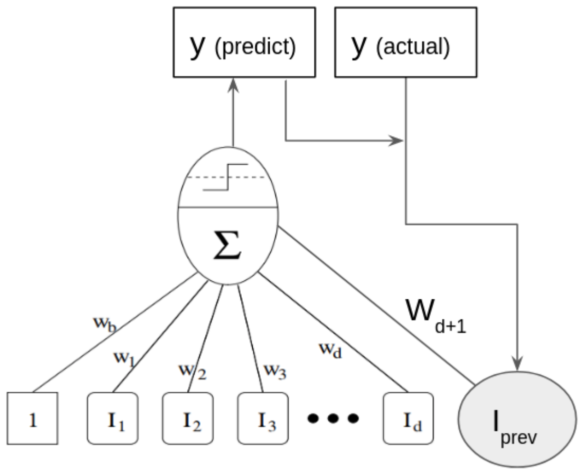

**Figure 5.** A RNN structure with a unit that represents the result of the immediate previous branching

Even though we discussed the fitness of RNN model in Section 2, we did not run workload with RNN. One may think that the internal state, $I_{prev}$, can be added in Figure 5, compared to Figure 2. But this is redundant as the result of previous prediction has been stored in one of $I_i$ ($1<=i<=d$), except that it introduces another weight, $w_{d+1}$. As the additional weight may hinder the convergence while training, we chose to skip the experiment with RNN.

**ii) Skewed Initialization**

We may reduce the misprediction rate if the weights of the unit are initialized with left-skewed normal distribution. It will nudge the predictions at the beginning to return positive value, meaning taken, which is much more likely than not taken. Only about 8.38% of branch is not taken in our workload. Or we may initialize the values with a snapshot after executing an enough number of instruction.

## 7. Conclusion

As the effective prediction of branching may well reduce overall CPI by minimizing control hazard, we explore the design space of prediction model with the extensive workload. The most powerful model armed with various predictors and 2-level prediction, showed 0.1782% of misprediction rate under unlimited budget. On the other hand, the simplest model, perceptron, showed 0.5631% with 32KB storage constraint. Based on the two results an architecture designer can foresee that the misprediction rate would be between them, and arbitrate the storage (or silicon area) for branch prediction and the clock frequency that would maximize CPI. Moreover, given that the perceptron is small and simple but quite effective at simple tasks, one may apply it to tasks under a highly time-constrained environment, like network measurement [27]. We made our implementation available at https://github.com/iriszero/NN-branch-prediction for future research.

## References

[1] CireşAn, Dan, et al. "Multi-column deep neural network for traffic sign classification." Neural networks 32 (2012): 333-338.
[2] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
[3] Hannun, Awni, et al. "Deep speech: Scaling up end-to-end speech recognition." arXiv preprint arXiv:1412.5567 (2014).
[4] Chen, Xiaozhi, et al. "Multi-view 3d object detection network for autonomous driving." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.
[5] Errico, James H., et al. "Collaborative recommendation system." U.S. Patent No. 8,949,899. 3 Feb. 2015.
[6] Shen, Dinggang, Guorong Wu, and Heung-Il Suk. "Deep learning in medical image analysis." Annual review of biomedical engineering 19 (2017): 221-248.
[7] Alsheikh, Mohammad Abu, et al. "Mobile big data analytics using deep learning and apache spark." IEEE network 30.3 (2016): 22-29.
[8] Wang, Yibo, and Wei Xu. "Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud." Decision Support Systems 105 (2018): 87-95.
[9] Monti, Federico, et al. "Fake News Detection on Social Media using Geometric Deep Learning." arXiv preprint arXiv:1902.06673 (2019).
[10] Keith, Katherine A., et al. "Identifying civilians killed by police with distantly supervised entity-event extraction." arXiv preprint arXiv:1707.07086 (2017).
[11] EurekAlert!. "Army researchers develop new algorithms to train robots". Retrieved 2019-06-13.
[12] Kramer, Adam DI, Jamie E. Guillory, and Jeffrey T. Hancock. "Experimental evidence of massive-scale emotional contagion through social networks." Proceedings of the National Academy of Sciences 111.24 (2014): 8788-8790.
[13] Vintan, Lucian N., and Mihaela Iridon. "Towards a high performance neural branch predictor." IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339). Vol. 2. IEEE, 1999.
[14] Vintan, Lucian. "Towards a Powerful Dynamic Branch Predictor." Romanian Journal of Information Science and Technology, Bucharest, Romania (2000).
[15] Jiménez, Daniel A., and Calvin Lin. "Dynamic branch prediction with perceptrons." Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001.
[16] Jiménez, Daniel A. "Fast path-based neural branch prediction." Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2003.
[17] McFarling, Scott. Combining branch predictors. Vol. 49. Technical Report TN-36, Digital Western Research Laboratory, 1993.

[18] Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." Neural computation 1.2 (1989): 270-280.
[19] Smith, James E. "A study of branch prediction strategies." Proceedings of the 8th annual symposium on Computer Architecture. IEEE Computer Society Press, 1981.
[20] Pan, Shien-Tai, Kimming So, and Joseph T. Rahmeh. "Improving the accuracy of dynamic branch prediction using branch correlation." ACM Sigplan Notices. Vol. 27. No. 9. ACM, 1992.
[21] Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." Neural computation 1.2 (1989): 270-280.
[22] Seznec, Andre. "Analysis of the O-GEometric History Length branch predictor." ACM SIGARCH Computer Architecture News 33.2 (2005): 394-405.
[23] Seznec, André, et al. "Design tradeoffs for the Alpha EV8 conditional branch predictor." ACM SIGARCH Computer Architecture News 30.2 (2002): 295-306.
[24] Seznec, André. "A case for (partially)-tagged geometric history length predictors." Journal of InstructionLevel Parallelism (2006).
[25] Seznec, André. "Tage-sc-l branch predictors." JILP-Championship Branch Prediction. 2014.
[26] Hagan, Martin T., and Mohammad B. Menhaj. "Training feedforward networks with the Marquardt algorithm." IEEE transactions on Neural Networks 5.6 (1994): 989-993.
[27] Yang, Tong, et al. "Elastic sketch: Adaptive and fast network-wide measurements." Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. ACM, 2018.