An Evaluation on a DNN-based Branch Predictor that References Branch Target Buffer

Abstract

With the increasing depth of Out-of-Order pipeline, and the introduction of speculative execution, it is becoming more pivotal to have an accurate branch predictor to hide the delay and the mis-prediction penalty from processors. In the past decade, researchers have proposed various types of Branch Predictors (BP) based on Machine Learning (ML). A machine learning model takes various of data as input, such as branch history, PC, etc. and outputs a single bit as a prediction on branch taken / not take. Currently no proposed model has included the parameter of *branch target*, which theoretically included certain amount information on the branch result, which would also contribute to the accuracy of the BP. An Evaluation on a BP model that included the *branch target* as parameter was made, and the accuracy of the model did not converge throughout the training due to various of reasons.

Background

As the depth of the out-of-order processor pipeline getting deeper, as well as the introduction of speculative execution, it is becoming more pivotal to have an accurate branch predictor to hide the delay and the mis-prediction penalty from processors. As conventional static or Look-up Table (LUT) based branch predictors are hitting the prediction accuracy bottleneck, some researchers are exploring the field of Machine Learning (ML) to seek an even more powerful solution towards the problem. A branch predictor, in nature, is a binary classifier which ML solutions excel at. In the past 20 years, scientists have proposed numerous of ML-based branch predictors. One of the tendencies of the proposed ML models on branch predictors was that the size, dimension, and the number of layers used in the ML models has been increasing; and the accuracy of the models are also increasing over years.

Machine Learning have become one of the most popular algorithms as a classifier. A classifier takes the description of a certain object's characteristics as input and output the identification group the object belons to. One of the most classical classifiers out there is handwriting recognition (HWR) algorithm, which a model takes in the pixel values of an image containing a single character of one's handwriting (a-z, 0-9), and output which character the image contains. In the application such as branch prediction, the output is either 0 (not taken) or 1 (taken). Such classifier is called a binary classifier.

Neural Network (NN) is one of the most commonly used model structures in Machine Learning. As shown in fig 1, the example neural network has 3 inputs and 2 outputs with one hidden layer. The hidden layers acts as a black box to the user and performs a weighted sum, offset, and roofing on all data on the input layer. Finally, the output layer's output represents the most likely classification for the input.

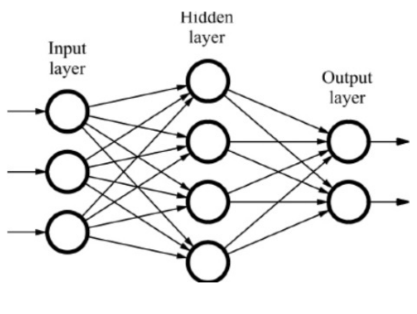


Fig 1, a typical, simple neural network with one hidden layer

Deep Neural Network (DNN) is the NN that have multiple hidden layers (fig 2), hence calling “deep” as the data travels through multiples stages of hidden layers. DNNs used to be impractical to use as its very computational costly in the past, but have been much more popular thanks to advanced computational hardware.

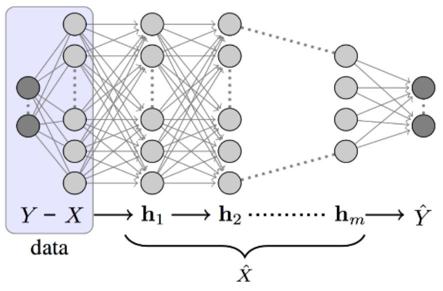


Fig 2, a typical deep neural network

Prior work and Methodology

The most recent work on ML based branch predictor involves merging several hardware accessible data into a single array composted of single bits. The data being merged included the current branch’s Program Counter (PC), Global Address (GA, which are the LSBs of prior branches), Global History (GHR, which are taken / not taken sequence of past branches globally), and Local History (LHR, similar to GHR but only on current thread). The data involved are merged and being fed into a black box model to be trained / verified as shown in fig 3. The data structure that’s being merged and compiled is shown in fig 4.

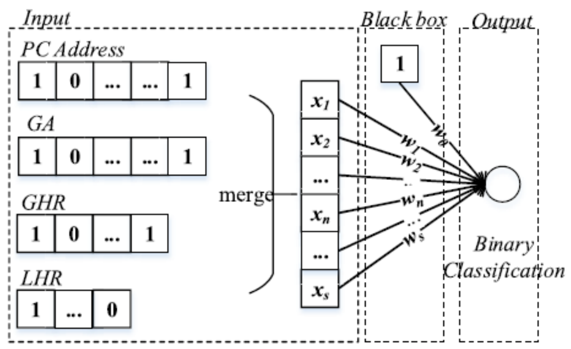


Fig 3, Branch Predictor classification model

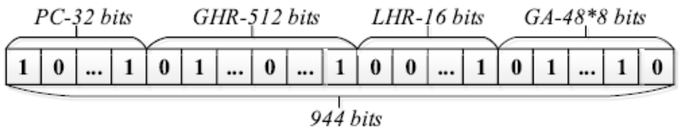


Fig 4, the input data structure into the DNN

The idea of machine learning is to feed in as much information into the model as necessary and find the relation among data points throughout the training process. The main idea and of the method proposed by this project is that, except the branch history and PC, there could other hardware-reachable information that may contains useful information on predicting the branch outcome such as the *branch target.* With the rather reasonable premise of the target of a certain branch can give certain amount of insight on the outcome of certain branch, it is possible worth to evaluate whether containing the branch target parameter into the data structure in the prior work and see the outcome. This premise intuitively holds when considering a loop program that often branches back to the start of the loop, which is a fixed 'branch target'. After adding 'branch target' variable, which is an instruction address of 32 bits, the input data is now 976 bits (fig 5)

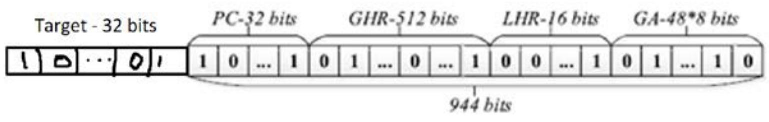


Fig 5, updated input data structure to DNN

It is worth mentioning that for indirect branches, the branch target is often can only be solved after the value of the source register is solved, and it is rather impractical to predict the branch after solving the register value. So, during actual deployment, the branch target variable is not referenced from register resolving, but from access branch target buffers (BTB) (fig 5). Based on a gem5 simulation on some regular tasks such as array sorting, it was observed that BTBs have a more than 90% hit rate. Although not necessarily all hits are actual branch buffers due to aliasing, but it should be enough for research propose as we only evaluate the accuracy when predicting a branch which is accessible from BTB.

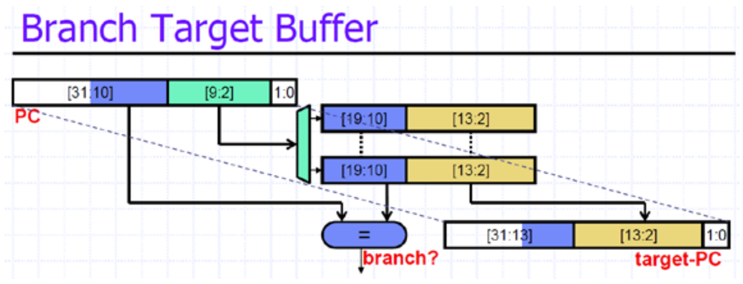
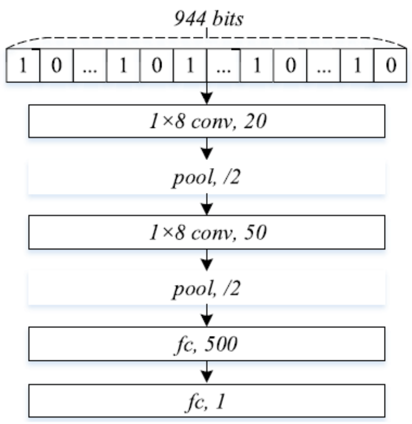


Fig 6, a typical BTB structure

The training data was extracted from running sorting benchmarks on a full system simulation (SE mode) in Gem5 simulator, during the benchmark, all *indirect branches* were recorded with their PC, target, and Taken bits logged. A total of 1 million entry of data were recorded, and the data were then fed into a trainable DNN model based on a TensorFlow. Several model structures were evaluated include some pre-defined classical structures introduced in prior work such as LeNet (fig 7) or AlexNet (fig 8), and the layer parameters fine-tuned to accommodate to the larger input data length.



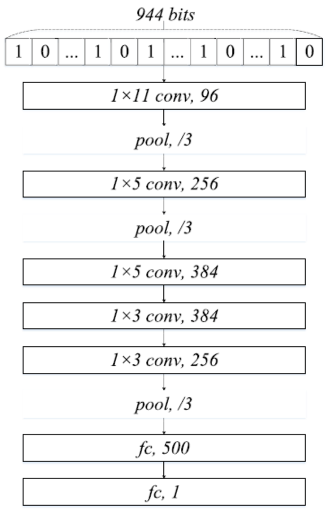
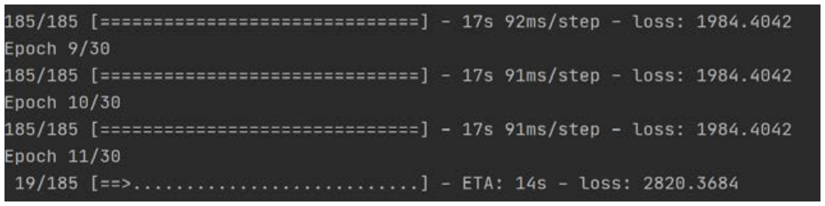


Fig 7, implemented LeNet structure.   Fig 8, implemented AlexNet (fig 8)

Results and analysis

Unfortunately, there is nothing much to show for the results part, as all model structure used in prior work failed to converge, no matter how the parameter was tuned, the trained branch predictor all converged at "predict all taken" (fig 9). There could be numerous of reasons on why a model failed to converge, but the most typical reason is that the relevance among input data points is very poor. This reason could be intuitively true. For example, when the computer sees the PC value say like 0x0000_0014, this value makes perfect sense for computer to interpret as a single value. However, when this PC value is decomposed to 32 discrete, single-bit value, the relevance among each bit is very poor. This could be the main reason why the model failed to converge.



```
185/185 [==============================] - 17s 92ms/step - loss: 1984.4042
Epoch 9/30
185/185 [==============================] - 17s 91ms/step - loss: 1984.4042
Epoch 10/30
185/185 [==============================] - 17s 91ms/step - loss: 1984.4042
Epoch 11/30
 19/185 [==>...........................] - ETA: 14s - loss: 2820.3684
```

Fig 9, training log shows the model failed to converge as the training loss is increasing among epochs.

Checking the output shows the model predicted all input as "taken" regardless the input value

It is worth notice that although the paper of the prior work included its training data structure and model shape, everything else was kept rather untransparent and the parameters I entered into the model is far from fine-tuned. Moreover, the training data set from the original paper was also not included so I could not evaluate the correctness of the original paper, not to mention the proposed change I made for this project.

Conclusion

ML based branch predictor are still competitive in accuracy under certain cases. However, data structure and input data selection are not everything that will guarantee to generate a perfect model. The training data set as well as fine-tuning the model also plays some major rolls. Due to the lack of transparency on the latter parameters, I was not able to produce an effective model for the branch prediction workload for unknown reasons. One of the future works I could do is definitely to look deeper into the parameter details that I can fine-tune within the model, and see if that will yield a converging result.

References

Y. Mao, H. Zhou, X. Gui and J. Shen, "Exploring Convolution Neural Network for Branch Prediction," in IEEE Access, vol. 8, pp. 152008-152016, 2020, doi: 10.1109/ACCESS.2020.3017196.

Supplement Material:

All materials are available in the GitHub link: Bald-Badger/DNN-Branch-Predictor (github.com)

Code and data in ~/python

reference in ~/reference

other proposal / report in ~/