Branch Prediction with Convolutional Neural Networks

Siyuan Chen, Frank Yin

December 7, 2018

Abstract

Following the work of Jimenez's Perceptron [1] predictor, this project aims to explore the performance of larger-neural-network-based branch predictors that are feasible to implement in real processors. Model is constructed and evaluated in Tensorflow from traces in Championship Branch Prediction 2016. Comparisons with two state-of-the-art branch predictors as well as across different models are presented. A detailed discussion of the results, which found the models to be performing poorly and some related future opportunities is also given.

1 Introduction

Control hazards have always been one of the most significant performance barriers in modern processors. To minimize the negative effects from control hazards, researchers have put decades of effort into branch prediction research. However, as accuracy of branch predictors keeps increasing and reaches over 98%, the final gap to 100% accuracy is proved increasingly harder to achieve.

Recently, machine learning has gained increasing popularity as it proves to be an effective tool to predict and classify images, search results, medical history and so on. Computer architects have mainly been granted the commercial opportunity to build special-purpose machine learning accelerators, but more and more researchers look into integrating machine learning to improve current microarchitecture implementations.

As a matter of fact, machine learning is not a completely foreign idea for branch predictors. In 2001, Daniel Jimenez and Calvin Lin in their paper *Dynamic Branch Prediction with Perceptrons*[1] proposed a scheme to employ the smallest neural network - the perceptrons to build a branch predictor. Each perceptron has weights that could be dynamically updated according to the previous branch histories. The paper opened up a whole new area of ideas for branch predictors, but still cannot correctly predict all the branches.

Based on the idea proposed by the perceptron paper, this paper looks into using more complex neural network models, namely Convolutional Neural Networks (CNN) to predict branches in a more accurate fashion while sacrificing complexity. This rest of this report will be organized as the following: Section 2 introduces previous work examined. Section 3 describes in depth the CNN models we tested. Section 4 presents the results and comparisons obtained from experiments and discusses the implications behind these data. Section 5 shares some insight about what possible related areas could be explored in the future and section 6 concludes the report.

2 Previous Work

In 2001, Daniel Jimenez in his paper [1] proposed using perceptrons to predict branches. The perceptrons are not pre-trained but start with weights biased towards the positive side, i.e. branch taken. Then, at run-time, the weights are trained with a very simple algorithm that involves only the use of multiplication and addition. When a branch instruction is fetched, it is fed into the net of perceptrons which, after calculation, would output either a positive or negative result indicating taken (positive) or not taken (negative).

With the advancing research of neural networks, Mao et al. [2] explored implementing larger scale neural networks in hardware as a branch predictor. They first examined the Deep Belief Network model and then examined two CNN models respectively based on LeNet and AlexNet models. Their results show an accuracy exceeding that of the Perceptron [1] branch predictor but not the TAGE [3] which is considered the best branch predictor currently.

However, all the models they explored have up to 10 layers and thus would pose a complexity challenge in realistic implementation.

After identifying this substantial problem in [2], this project aim to explore the possibility of implementing neural networks in branch predictors in a more realistic fashion by shrinking the size of the neural network models. The results are evaluated and compared with the perceptron and TAGE predictors in the following way:

A neural network model will be constructed using Tensorflow in Python. The training data used comes from the training traces in Championship Branch Prediction 2016 (CBP2016). However, due to the large size of traces provided, the authors don't have the computing power to train the model with all the data provided and have to randomly choose 10 traces from all the four categories provided: Short Mobile, Short Server, Long Mobile, Long Server. The evaluation data also comes from CBP2016's evaluation data and the results are compared to evaluation results in CBP2016.

3 Neural Network Models

The following section will describe more in depth the CNN models employed and the reasons behind their construction.

To discuss the choice of these models, an explanation of why CNN is suitable for branch prediction is needed. CNNs are most often used to process image data and classify images. They have the advantage of being able to exploit the locality across pixels of an image. Previous research in branch predictors almost all exploit the locality in branch directions by using branch history tables or registers. Due to this same abundance of locality, CNN seems like a natural choice for applying machine learning in branch prediction.

In [2], several models with 8 to 10 layers are explored in depth. These models, however, would have introduced complexity and latency both unsuitable for most realistic applications. Thus, in this project, the models only have 1 or 2 "layers" (convolution + pooling), so that the possibility of applying neural networks in branch prediction could be explored in a

realistically feasible manner.

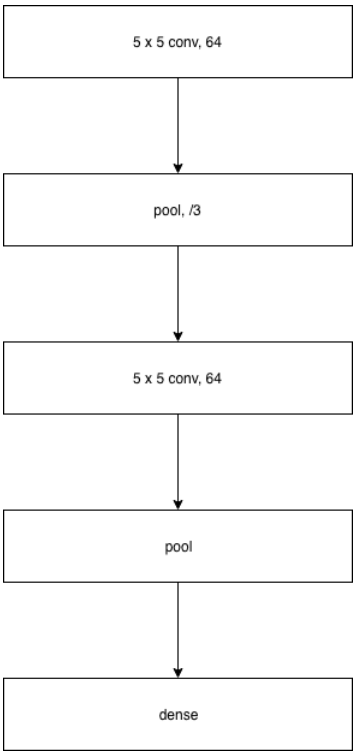

Figure 1: CNN1 layers

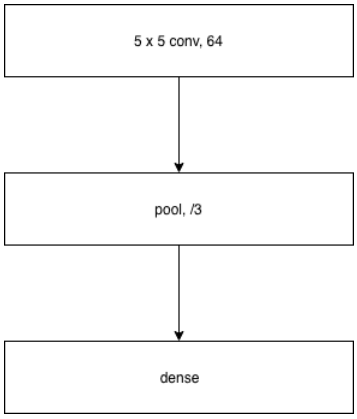

Figure 2: CNN2 layers

3.1 CNN1

As shown in Figure 1, CNN1 has two convolutional layers *conv1* and *conv2*. Both of them have kernel size of [5, 5] and 64 filters. After each convolutional layer, the data is input into a pooling layer to reduce size and avoid overfitting.

3.2 CNN2

Compared to CNN1, CNN2 has only one conv + pooling layer.

4 Results and Discussion

4.1 Comparison with state-of-the-art predictors

After both models are trained with all the randomly chosen traces, four traces, one from each category in CBP2016's evaluation traces are chosen. The evaluation result (using accuracy as the only metric) is then compared to Andrew Seznec's LTAGE [3] 8KB submission and Daniel Jimenez's Perceptron [1] 8KB submission. The results are shown in Table 1.

Predictor	LONG MOBILE-32	LONG SERVER-4	SHORT MOBILE-3	SHORT SERVER-116
LTAGE	0.989	0.988	0.996	0.987
Perceptron	0.988	0.985	0.989	0.974
CNN1	0.664	0.515	0.901	0.504
CNN2	0.658	0.511	0.898	0.469

Table 1: Accuracy comparison between four branch predictors

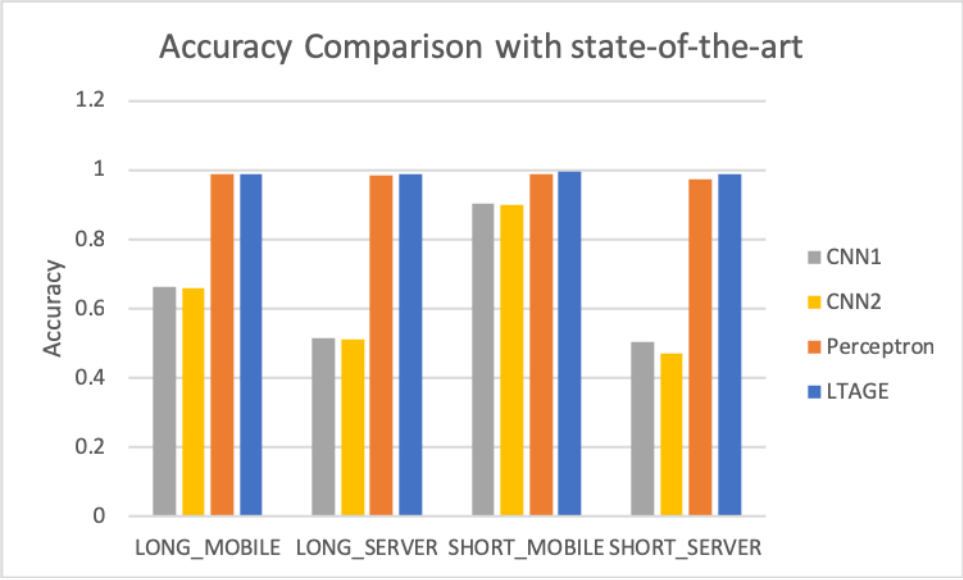

Figure 3: Accuracy comparison between four branch predictors

From the data shown that, we could see that the state-of-the-art branch predictors outperforms CNN1 and CNN2 in all benchmarks we explored. For Long-Mobile, Long-Server,

Short-Server, the CNN1 branch predictor is performing even worse than a default guess of taken, which statistically would have a 80% accuracy.

The explanation we give behind this behavior is that although for a single loop, the address behavior is very similar, by statically studying loops that could vary in target PC by a lot, the shallow CNN is constantly being "driven" to the other direction just as we saw in 1-bit saturating counters. Apparently, according to [2], a deeper CNN would solve this issue by a huge amount, but complexity is also sacrificed. This could also be shown as CNN1 outperforms CNN2.

4.2 Special purpose Processors

After realizing the complexity of a branch predictor based on neural networks, a possible application of these branch predictors would be on special purpose machine learning accelerators that already have the computational units used in the neural networks implemented on chip. To show this, a predictor specifically trained for one category of data (SHORT MOBILE) is evaluated across all four categories in Table 2.

Predictor	LONG MOBILE-32	LONG SERVER-4	SHORT MOBILE-3	SHORT SERVER-116
CNN1	0.664	0.515	0.901	0.504
CNN2	0.658	0.511	0.898	0.469

Table 2: Accuracy comparison for a predictor specifically trained for short mobile

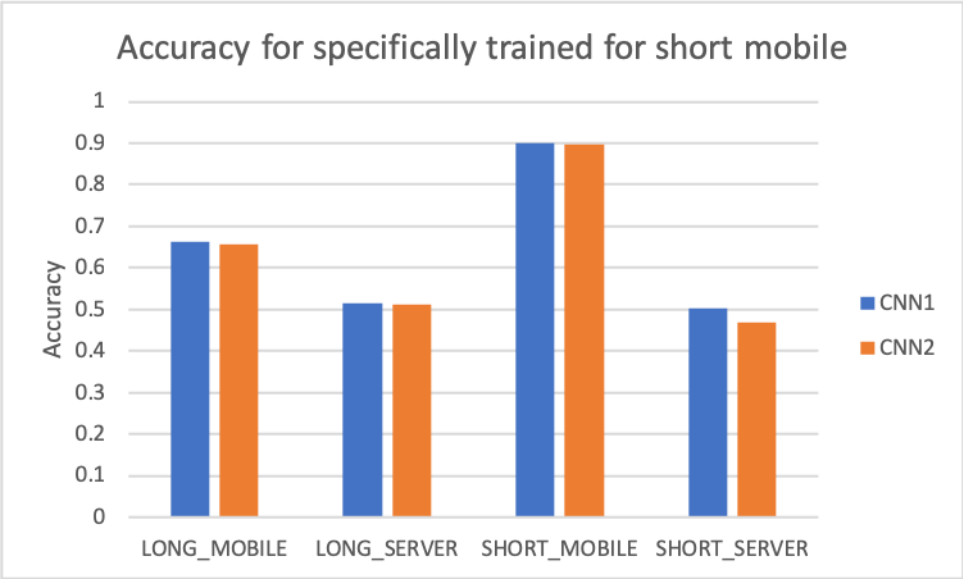

Figure 4: Accuracy comparison for a predictor specifically trained for short mobile

From Table 2 and Figure 4, we indeed see that the model performs much better in SHORT MOBILE trace evaluation than the other three categories, but because of our limited data points taken, we refrain to conclude that a specifically trained branch predictor would perform better in the category that it is trained in.

In addition, even if we could conclude neural network branch predictors perform better in certain special purpose programs, some of these programs such as machine learning, has a lot of data level parallelism that could be exploited with SIMT or SIMD machines where branch prediction may not even be needed.

It is easily observed that the data we obtained here with only SHORT MOBILE training is the same as the data in the previous section 4.1 when predictors are trained in all four categories. This behavior will be discussed in Section 4.3.

4.3 Size of Pre-training Data

As our predictors are all pre-trained, the commercial semiconductor companies, with access more program traces and benchmarks, could potentially come up with a much better trained model. Here, we try to validate this behavior by comparing CNN2 trained in 1 step, 3 steps and 5 steps where each step contains equal size data.

Predictor	LONG MOBILE-32	SHORT MOBILE-3
CNN2-1	0.664	0.901
CNN2-3	0.664	0.901
CNN2-5	0.658	0.898

Table 3: Accuracy comparison for a predictor trained with different size of data

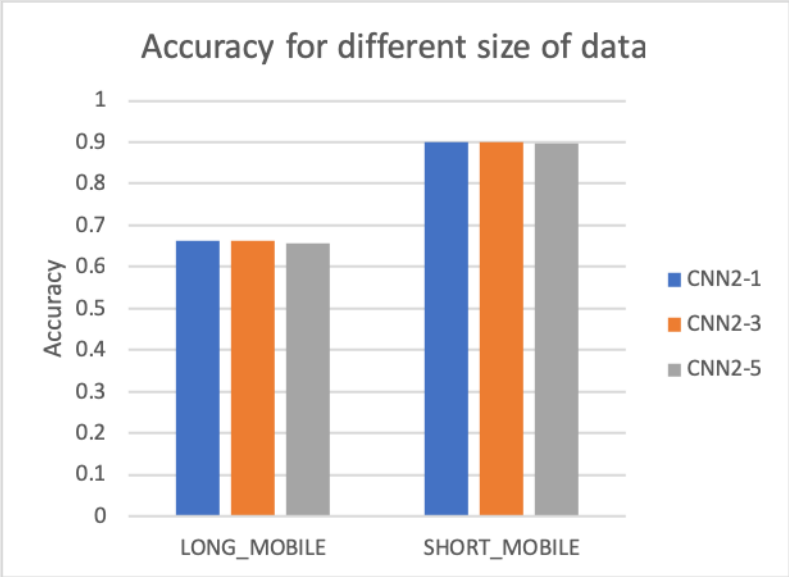

Figure 5: Accuracy comparison for a predictor trained with different size of data

The accuracy is not increasing as we expected it to but stayed almost unchanged. One explanation is that the data size for each step is too small to observe a jump in evaluation accuracy.

In addition, we notice that parts of our data is exactly the same as the accuracy evaluated using more data in all four categories. After putting in some thoughts, we think that one possible reason behind this unexpected outcome is that our step size is set too low so that the model could be saturated easily. However, lacking the computing power to test models with large step sizes, we couldn't continue to verify our theory.

5 Future Efforts

5.1 Dynamically Trained Neural Network Algorithms

One of the biggest reasons why Perceptron branch predictors proved to be a success while CNN1 and CNN2 behaved poorly is that perceptrons are dynamically trained (weights updated). Currently, dynamically training algorithms for neural networks are too complicated to implement in hardware for a branch predictor. However, with increasing research efforts into neural networks. An efficient dynamically training algorithm could improve the performance of neural-network-based branch predictors by a lot.

5.2 Using Machine Learning To Mitigate Spectre

The L1 cache side-channel attacks Spectre and Meltdown exploit speculative executions. Especially, spectre attacks intentionally trick a branch predictor into predicting "taken" for a branch (usually a bounds check) that is not supposed to be taken. One way to mitigate spectre attacks would be improving branch predictors so that they would not be tricked in important security checks such as array bounds check. Future researchers could look into using machine learning in branch predictors so that they would not be fooled in these bounds-checking branches.

6 Conclusion

In this project, we looked into employing larger neural networks than Daniel Jimenez's perceptron into a branch predictor while keeping the depth of these neural networks feasible for actual implementation.

After evaluation and comparison with state-of-the-art branch predictors, we found that the models perform poorly in most benchmarks and could not exceed the accuracy of state-of-the-art branch predictors for all benchmarks. We concluded that insufficient training may cause the ill performance, but employing neural networks in branch predictors would not be

optimal in most general purpose CPUs.

References

[1] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons", ACM *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001

[2] Y. Mao, J. Shen and X. Gui, "A Study on Deep Belief Net for Branch Prediction," in IEEE *Access*, vol. 6, pp. 10779-10786, 2018.

[3] A. Seznec, "Exploring branch predictability limits with the MTAGE + SC predictor", *5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction, ISCA-43*, June 2016.