University of Central Florida

Department of Computer Science

CDA 5106: Spring 2023

Machine Problem 2: Branch Prediction

Branch Prediction Using Neural Networks

by

Group 1

Honor Pledge: ”I have neither given nor received unauthorized aid on this test or assignment.”

Student’s electronic signature: Alina Ageichek
Student’s electronic signature: Brayan Villier
Student’s electronic signature: Daniel Cisneros
Student’s electronic signature: Dhruvin Kamani
Student’s electronic signature: Munaimah Khan
Student’s electronic signature: Shehreen Azad

Branch Prediction Using Neural Networks

I. Introduction

Branch prediction is an imperative approach to maximizing parallelism and processor performance. Along with other forms of parallelism like instruction level parallelism and instruction cycle pipilining, branch predictions aims to pre-prepare the branch instruction. The goal of branch prediction is to decide the direction the branch address will take. Dynamic branch prediction depends on the branch history, predicting between taking or not taking a branch based on the prediction direction and behavior of previous branched [1]. A counter indicates the prediction direction which is updated based on the mispredictions or correct predictions. Due to this self-learning behavior, branch prediction can be simulated with deep learning principles such neural networks. Neural networks are a connection of processors ("neurons") that sense environment variables, taking in the input and relearning to create more accurate output results [2]. Using the underlying self learning motivation behind neural networks and the analogous predictive methodology of branch prediction, accuracy can be tested between baseline simulator methods (gshare, bimodal, hybrid, and smith) and NN-based simulators including perceptron, convolution neural network, recurrent neural networks, and multi-layer perceptron. All baselines and NN simulators were run on three individual trace files and tested for prediction accuracy, i.e. misprediction rates were compared for all simulators. The prime comparison between simulator accuracy is tested through misprediction rates. All simulators were implemented in python and graphical analysis is explained in the conclusion.

This report is organized as follows: Section 2 provides a literature review of previous approaches to branch prediction. Section 3 describes our proposed approach with detailed description of baseline and neural networks' designs. Section 4 presents our experimental setups, evaluation and lastly a discussion of the results. Finally, Section 5 provides our conclusion.

II. Literature Review

Branch prediction has been the subject of extensive study and application in both academia and the industry since it plays a crucial role in many contemporary pipelined microprocessor architectures in reaching great performance. Problems of branch predictions have been carefully investigated by many researchers and they came up with different solutions for the same. The two-level scheme introduced by Yeh and Patt (1991) [3] was a game change back in the day since they introduced a branch prediction scheme used in computer architecture to improve the performance of pipelined processors. Their approach uses a mix of branch address and global or per-branch history to index a pattern history table (PHT) of two-bit saturating counters. The forecast is determined by the high bit of the counter. The number is increased if the branch is taken and decremented in the opposite case once the branch conclusion is known. Aliasing is a significant issue with two-level predictors [4], and many of the proposed branch predictors [5]–[8] aim to solve the issue while maintaining the fundamental prediction process.

Two-level predictors that are out that cannot consider longer history lengths, and this poses a problem when the distance between correlated branches exceeds the length of global history shift register [9]. It would not help even if a pattern history table scheme somehow implemented longer history lengths since it would require higher training time as the history length is longer for these types of schemes [10]. Another scheme for considering longer paths is Variable length branch prediction [11]. It computes a hash function of the address along the path of the branch which eliminates the capacity problem of the pattern history table. This achieves great performance since it considers longer histories, but it is impractical for a real architecture because it uses a complex multipass profiling and compiler feedback mechanism.

In contrast, machine learning has been applied to branch prediction in order to improve the accuracy of branch predictions beyond what is possible with traditional branch prediction schemes [12]. Jimenez et al. (2001) [13] proposes a neural predictor which is based on a basic neural network called Perceptron [14], where they represent the forward inference process of branch prediction by using a layer of the Perceptron model. Jimenez et al. (2003) [15] proposes a path-based prediction algorithm with a higher prediction accuracy and some further improvements. In 2005, Jimenez proposes a new algorithm namely a Piecewise-Linear Prediction algorithm [16] and this optimizes the prediction accuracy of the linear non-separable function.

III. Proposed Approach

In the following sections, we describe the details of the implementation of baseline methods such as the Smith, Bimodal, Gshare, and Hybrid branch predictors, and some neural network-based methods such as the Single-layer Perceptron, the Multi-layer Perceptron (MLP), 1-D Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

A. Baselines

Smith, Bimodal, Gshare, and Hybrid branch predictors are considered as the baseline models and they are discussed in

Sec. III-A1 through Sec. III-A4.

1) Smith: This is one of the oldest proposed dynamic branch predictors. Fig. 1 shows a basic n-bit Smith predictor. Smith algorithm has a table that keeps a record of the outcomes for each of the branches whether each branch was taken or not. The table consists of 2n counters in which the counter keeps track of the previous branch's direction. Only 2n entries exist, thus the program counter (PC) is hashed down to n bits. The width of each counter in the table is k bits. The branch direction prediction uses the MSB of the counter. If the MSB is 1, it is expected that the branch will be taken; if it is 0, it is predicted that the branch will not be taken. The counter is updated based on the branch outcome once the branch has been resolved and its real direction is known. The counter is only increased if the branch was taken and the current value is less than 2k-1. If the branch was not taken, a counter whose value is larger than 0 is decremented. The saturation counter won't be thrown off by a single aberrant decision as a result of considering the histories of numerous recent branches because there are more history bits acting as inertia. 2-bit counters offer higher prediction rates than 1-bit counters for tracking branch directions. However, adding a third bit only slightly enhances performance, and in most designs, this slight advantage is not worth the 50% increase in space required to accommodate the third bit.

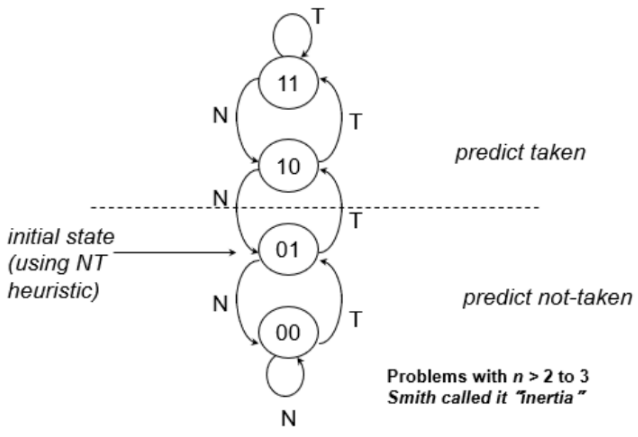

Fig. 1: Basic n-bit Smith predictor

2) Bimodal: This branch prediction technique is a simple dynamic branch prediction that uses 2^n entry bits which uses the lower bit of the Program counter (PC) or branch history address [17]. This design follows the simple diagram shown in figure x where the index is formed from the lower PC bits. As shown in Fig. Fig. 2, the lower bit of the PC is indexed(m) to a 3-bit counter that predicts whether a branch will be taken or not. If for example, a bit is indexed to a non-taken branch in the 3-bit counters, two possible outcomes would occur: if the actual branch outcome is taken and the counter's prediction is not, a misprediction will occur. The latter will also emerge if the counter's prediction shows that a branch is taken whereas the actual branch outcome is not.

Bimodal is implemented by first creating a prediction table that contains a 3-bit counter. All entry values are initially 4, which represents weakly taken. Following counter logic, any value between 0 and 3 are not taken whereas any value between 4 and 7 are considered taken for branch prediction. Next a branch PC index is calculated by right shift the address given by 2 and mask it with total size. Total size is 2^m where m is the number of PC bits used to index the bimodal table. The branch's index is used as a representative of the branch's counter from the prediction table. If the value of the counter is greater or equal to 4, taken is predicted or the opposite occurs. The prediction counter is then updated by increasing its counter by 1 if the branch is "taken", or it will be decremented.

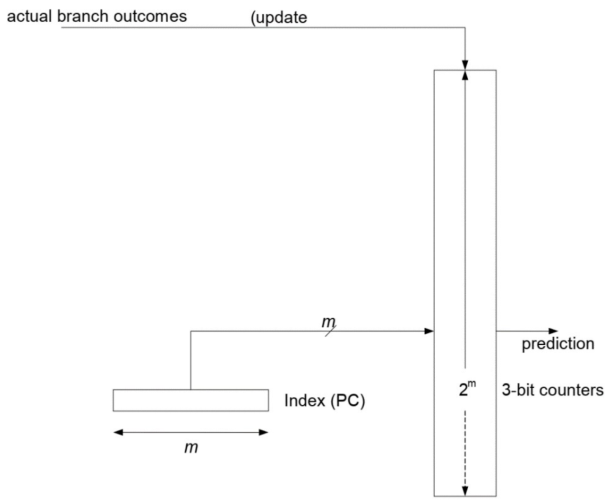

Fig. 2: Bimodal predictor

3) Gshare: One of the commonly used techniques for branch prediction is the Global History Register (GHR)-based GShare predictor. The GShare predictor exploits the correlation between the current branch instruction's behavior and its past behavior to predict whether the branch will be taken or not. It maintains a GHR, which is a register containing a history of the recent branches executed, and uses it to index a table of counters. The table of counters contains a prediction for each history pattern.

The gshare simulator involves the following components: the global history register (GHR), the prediction table (P-Table), the PC Address, and the M and N parameters. This gshare simulator uses a 3-bit counter that saturates on the two extremes (0 and 7). As seen in the bimodal, counter values in the range 0 - 3 (inclusive) indicate a not taken prediction direction and 4 - 7 (inclusive) indicate a taken prediction direction. The P-Table is indexed from 0 to 2^M - 1 and all indices are initialized to weakly taken - the number 4. PC addresses and prediction values are read in from global history. When N ¡ M, the GHR is appended with 0's so that the GHR and the least significant M bits of the PC address undergo an XOR operation, yielding the correct index in the P-Table for the current address. Based on the prediction direction, the value at the index is updated and either a prediction or misprediction occurred. The GHR reflects this as a left bit shift and appending a 1 or 0 for prediction or misprediction, respectively.

4) Hybrid: Hybrid branch prediction, as the name implies, is a known technique that makes a prediction based on the state of two or more branch prediction techniques [18]. Our design focuses on a simple hybrid that chooses between Bimodal and Gshare branch prediction. As Tab. I indicates, if both Bimodal and Gshare are correct or incorrect, no action is needed on the hybrid predictor. However, if either one is in the opposite state, for example, if Bimodal is correct while Gshare is not, then the Hybrid counter is incremented past the implemented threshold, assuming that is the very first prediction. As stated above, our hybrid , in this project, selects between the bimodal and gshare predictors. It is implemented with a chooser table 2^k 2-bits counters, where K is the number of PC bits used to index the chooser table. First, all counters in the chooser are initialized to 1. then one prediction is obtained from both bimodal and gshare from the trace file. Next, the branch index into the chooser table is determined by having the index as bit K+1 to 2 of the branch PC.
Once a prediction is made, index is used to to get to the branch's chooser counter. If chooser counter value is greater than 2, then we use the prediction from gshare. Otherwise, we use the prediction from bimodal. The actual branch predicts the outcome of branch chooser. The latter is increased if the former is taken, else it will be decremented. Only the branch predictor that was selected is updated. The gshare's global branch history register is always updated regardless if Gshare or bimodal is selected. The chooser table is then incremented or decremented based on the policy table indicated on the implementation section.

B. Neural Networks

In computer architectures, branch prediction is a technique used to improve the performance of CPUs by predicting the outcome of conditional branches. Therefore, one approach to obtain better results in branch prediction outcomes will be by using neural networks.
Neural networks are a type of machine learning model that can easily learn to recognize patterns and make predictions on different types of data. There has been a wide range of applications for neural networks, such as pattern recognition, classification, and image understanding. Branch prediction is not an exception to not applying it.

Bimodal	Gshare	**Action** on hybrid counter/chooser policy
Correct	Incorrect	Decrease
Incorrect	Correct	Increase
Correct	Correct	No action
Incorrect	Incorrect	No action

TABLE I: Action on chooser policy based on Bimodal and Gshare state

1) Single Layer Perceptron: Perceptron was introduced by Frank Rosenblatt in 1957 based on the original McCulloch-Pitts Neuron (MCP). The algorithm is mainly used for supervised learning of binary classifiers.
This algorithm has the advantage in the decision-making process and can be easily understood by examining their weights and the examination of these weights can lead to correlations that they have learned. While other techniques may not always be accurate in the decision-making, Perceptrons do not suffer from this issue since the decision-making process can be explained from a simple mathematical formula.
A single-layer Perceptron is the most basic form of the model. By single layers, it means that the model is composed of one artificial neuron connecting multiple input units to one output by weighted edges. From the mathematical form of Perceptron $y = w_0 + \Sigma_{i=1}^{n} x_i w_i$ the variables represent the structure of the model using x_i as the global branch history register but x_0 is always set to 1 to provide a "bias" input. Then the output y is calculated as the dot product of the weights w_i and input x_i which are both vectors. Fig. 3 shows the Perceptron model and how the input values are propagated through the weighted connections to compute the product and output the value y.
After obtaining the Perceptron output value, the following instructions are used to decide the training. If we let y be 1 if the output is taken and -1 if the branch is not taken. Then, let the threshold be the parameter that decides if enough training has been done. The instruction sets that if the branch outcome

```
$if\ (y_{output} \neq t)\ or\ |y_{output}| \leq \theta\ then$
    $for\ i := 0\ to\ n\ do$
        $w_i := w_i + tx_i$
    $end\ for$
$end\ if$
```

does not match the input x_i it will have a low correlation meaning that it will decrease the weights and increase the i^{th} when otherwise. The model was put on training for different iterations and threshold values. Finally, for the purpose of this project, we counted the number of correct and incorrect predictions to obtain the miss prediction rates as well as the accuracy of the model.
The only challenge for using the Perceptron model against the baseline was the time execution. The model takes more time to execute for every iteration that it makes, even though it gives one of the highest accuracies time execution is downward.

2) Multi-Layer Perceptron (MLP): The MLP model consists of multiple layers of neurons, with each layer

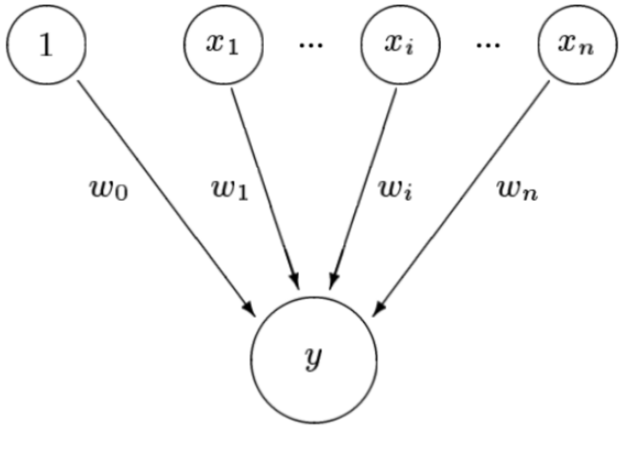

Fig. 3: Perceptron model's value propagation

performing a linear transformation of its input followed by a nonlinear activation function. The number of neurons in each layer, as well as the choice of activation function, can be tuned to optimize performance. To train MLP, a dataset of branch instructions and their outcomes is needed. The dataset is split into training and validation sets, and the model is trained using gradient descent to minimize a loss function that measures the difference between the predicted and actual outcomes.
Our proposed MLP was given the input data of trace files and read the taken or not taken branches in terms of 0s and 1s to compute the miss prediction rate and accuracy. The model was trained for 25 epochs, containing three hidden layers, 32 neurons each, and activation function as ReLU.

3) 1-D Convolutional Neural Network (CNN): Convolutional neural networks or ConvNets are part of modern machine learning that can handle data that are structured as numerous arrays. There are four principles behind ConvNets that leverage the inherent characteristics of natural signals, including local connections, shared weights, pooling, and multi-layers. The architecture is based on the fact that local groups of input values in array data like images are highly correlated that can be easily detected. The first few stages are composed of two types of layers which are based on convolutional layers and pooling layers. The role of pooling layers is to merge similar features, reducing the representation's dimensionality and creating invariance to small shifts and distortion.
Our proposed 1D CNN combines ConvNets and branch prediction to improve the performance of processors by obtaining a considerably better accuracy in terms of prediction and conditional branches. The model was given the input data of trace files and read the taken or not taken branches in terms of 0's and 1's to compute the miss prediction rate and accuracy. The model was trained for 25 epochs, containing three hidden layers, 32 filters and a kernel size of 3, and an activation function as ReLU.

4) Recurrent Neural Network (RNN): To use Recurrent Neural Networks (RNN) for branch prediction in computer architecture, the RNN is trained on the past history of branches and their outcomes to predict the outcome of future branches. The input to the RNN is a sequence of past branch outcomes, and the output is a prediction for the next branch outcome. The RNN is typically trained using supervised learning with a labeled dataset of branch histories and their corresponding outcomes. Once the RNN is trained, it can be used to predict the outcome of branches in real-time based on their past history. The accuracy of the predictions can be evaluated by comparing the predicted outcomes with the actual outcomes.
Our proposed RNN was given the input data of trace files and read the taken or not taken branches in terms of 0's and 1's to compute the miss prediction rate and accuracy. The model was trained for 25 epochs, containing three hidden layers, 32 units each, and an activation function as TanH. We tried to use ReLU for activation as it traditionally performs better than TanH, however, we found that TanH is the better-performing activation. One advantage of TanH over ReLU is that it is a bounded function, which means that its output is always between -1 and 1. This can help prevent large outputs from the network, which could cause issues like exploding gradients. In contrast, ReLU is an unbounded function and can produce large outputs, which could lead to instability during training. Since the RNN we used is not too deep, using ReLU was causing some overfitting issues. Hence, we opted to use TanH as the activation.

IV. Experiments

In this section, we evaluate the effectiveness of our proposed approaches against the baseline models. We provide an overview of the experimental setup and dataset to implement our approach in Sec. IV-A, we show the performance of the baselines, the neural networks and the comparison between the neural networks and the baselines in Sec. IV-B, and finally, we comment on the results of our experiments in Sec. IV-C.

A. Experimental Setup

All experiments were done using two local machines with Apple M1 and Apple M2 chips. All models are trained and evaluated across all three traces, namely, JPEG, GCC and Perl. The baselines (Smith, Bimodal, Gshare and Hybrid) and the Perceptron model were evaluated in terms of misprediction rate, whereas the multi-layer neural networks (MLP, CNN, RNN) were evaluated in terms of accuracy, F1 score, precision and recall. Now since the baselines are not neural network based approaches, it does not have concepts of confusion metrics and hence it is not possible to compare the neural network-based approaches against the baselines. Hence to compare the neural networks' performance against the baseline, we selected the misprediction rate metric. The neural networks have been trained with the history parameter varied across 4, 9, 15 and 21 and these values have been empirically set. The Smith predictor is evaluated with b = 1, 2, 3, 4, 5 ,6. For Bimodal, Gshare and Hybrid the m value is varied across 7, 8, 9, 10, 11, 12 and for Hybrid and Gshare n value is varied across 2, 4, 6, 8, 10, 12. For the multi-layer neural networks (CNN, MLP, RNN) the trace data was split into a 70%/15%/15% for training, validation, and test. We used Adam optimizer to compute the learning rates of 0.0001 for each parameter to update the weights and biases of the neural network during the training process. We also used the Binary Cross Entropy Loss as a loss function to measure the difference between the predicted true probability and the distribution of the target.

B. Evaluation

Fig. 4 through Fig. 7 shows the misprediction rate of the Smith, Bimodal, Gshare and Hybrid models across all three traces. From Fig. 4 through Fig. 7 we can conclude that the misprediction rate of the baseline model decreases as the m value increase. On the other hand, from Fig. 8 for the multi-layer neural network based models, we see that as the model looks at more of the previous history, the misprediction rate changes. This denotes the effectiveness of using more historical data for accurate predictions of the neural network based models. However, in case of the perceptron, the misprediction rate remained same across varying history parameters. This is due to the fact that since the perceptron is only a single neural network unit, changing the history does not affect its performance much as it does not learn more in the deeper layers based on the previous history. Tab. II shows the performance comparison of different models in terms of misprediction rates and it also shows the superiority of Perceptron over all other models across all three traces. Fig. 8 through Fig. 11 respectively shows the accuracy, f1 score, precision-recall of the neural network based models across all three traces. RNN is the best performing model across all those different evaluation metrics.

C. Discussion of Results

From the misprediction rates' graphs of different baseline models, it is clearly seen that the hybrid model has better performance than all of the other baselines. The bimodal predictor uses a single bit to keep track of branch direction history, while the GShare predictor uses a global history register to keep track of branch history. The hybrid predictor combines these two predictors to achieve better accuracy than either of them alone. On the other hand, RNNs are able to model the dynamic behavior of the program and capture long-term dependencies between the branch instructions by using its built-in memory mechanism. 1D CNN and MLP operate on fixed-size windows of the input and produces fixed length outputs. They do not have a built-in memory mechanism, which makes it difficult to model temporal dependencies. From Tab. II it is seen that single layer perceptron is the best performing model across all three trace files. A single-layer perceptron is better than multi-layer neural networks for small datasets because a single-layer perceptron is simpler and has fewer parameters, making it less prone to overfitting. In small datasets, overfitting is a common problem when using deep neural network models like MLP, RNN and 1D CNNs that have many parameters. The single-layer perceptron, on the other hand, has only one layer of weights, and thus a much smaller parameter space, making it easier to train on small datasets and less prone to overfitting. Additionally, the simpler structure of a single-layer perceptron allows for faster training and inference times, which is also advantageous in computer architecture applications where speed and efficiency are critical.

From Tab. II it is seen that the baseline models (except Smith) are performing better than the multi-layered neural network based models. Traditionally deeper neural networks have better performance than shallower networks, however, the performance of neural networks are quite sensitive to the quality and quantity of the training data. They require a large amount of training data and computational resources to perform well. In the case of branch prediction, this means that a large number of branch outcomes need to be collected and labeled, which can be difficult and time-consuming. If the training data is biased or limited, the neural network may not generalize well to new data and may not perform as expected, which is exactly what happened in our multi-layered neural networks. Moreover, this can be a challenge for hardware implementations that need to make fast and efficient predictions with limited resources. Secondly, neural networks are often considered black box models, meaning that it can be difficult to understand how the model is making its predictions. This lack of interpretability can make it difficult to diagnose and correct errors in the model. Lastly, the implementation of a neural network-based predictor may introduce additional design complexity and potential overhead, which may not be desirable in some scenarios where simplicity and efficiency are more important than accuracy. Hence the other approaches, such as hybrid branch predictor and simple linear models like Perceptrons, are computationally efficient and still provide good accuracy.

Model	Misprediction rate		
	GCC	JPEG	Perl
Smith	41.33	27.94	46.74
Bimodal	12.3	7.49	6.16
Gshare	12.53 (N=2)	7.30 (N=6)	6.42 (N=8)
Hybrid	10.16 (N=6)	6.98 (N=4)	4.16 (N=6)
Perceptron	**4.98**	**6.21**	**2.07**
MLP	41.42	10.13	2.72
CNN	61.61	14.99	20.63
RNN	34.43	9.70	2.66

TABLE II: Misprediction rate comparison

V. Conclusion

In conclusion, we have explored the use of neural networks for branch prediction in computer architecture. Through our experiments, we found that the hybrid model outperformed the traditional approaches of GShare, Bimodal, and Smith Predictor. Additionally, we found that the single layer perceptron outperformed other neural network architectures such as CNN, RNN, and MLP. This suggests that for small datasets, a simpler model such as the single layer perceptron can be more effective than more complex architectures. Lastly, we found that the single layer perceptron even outperformed the hybrid model. These results provide insights into the effectiveness of neural network-based approaches for branch prediction in computer architecture and offer guidance for future research in this area.

References

[1] D. R. Parthasarathi, "Computer architecture," Jul 2018. [Online]. Available: https://www.cs.umd.edu/ meesh/411/CA-online/chapter/dynamic-branch-prediction/index.html: :text=dynamic%20branch%20prediction.,branch%20be%20taken%20or%20not

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[3] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th annual international symposium on Microarchitecture*, 1991, pp. 51–61.

[4] S. Sechrest, C.-C. Lee, and T. Mudge, "Correlation and aliasing in dynamic branch predictors," *ACM SIGARCH Computer Architecture News*, vol. 24, no. 2, pp. 22–32, 1996.

[5] S. McFarling, "Combining branch predictors. tech. rep. tn-36m," *Digital Western Research Lab*, 1993.

[6] C.-C. Lee, I.-C. Chen, and T. N. Mudge, "The bi-mode branch predictor," in *Proceedings of 30th Annual International Symposium on Microarchitecture*. IEEE, 1997, pp. 4–13.

[7] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, "The agree predictor: A mechanism for reducing negative branch history interference," in *Proceedings of the 24th annual international symposium on Computer architecture*, 1997, pp. 284–291.

[8] A. N. Eden and T. Mudge, "The yags branch prediction scheme," in *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 1998, pp. 69–77.

[9] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, "An analysis of correlation and predictability: What makes two-level branch predictors work," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998, pp. 52–61.

[10] P. Michaud, A. Seznec, and R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," in *Proceedings of the 24th annual international symposium on Computer architecture*, 1997, pp. 292–303.

[11] J. Stark, M. Evers, and Y. N. Patt, "Variable length path branch prediction," in *Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, 1998, pp. 170–179.

[12] S. Mittal, "A survey of techniques for dynamic branch prediction," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4666, 2019.

[13] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 197–206.

[14] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[15] D. A. Jiménez, "Fast path-based neural branch prediction," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 2003, pp. 243–252.

[16] ——, "Piecewise linear branch prediction," in *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 2005, pp. 382–393.

[17] I. Bate and R. Reutemann, "Efficient integration of bimodal branch prediction and pipeline analysis," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*. IEEE, 2005, pp. 39–44.

[18] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 2019.

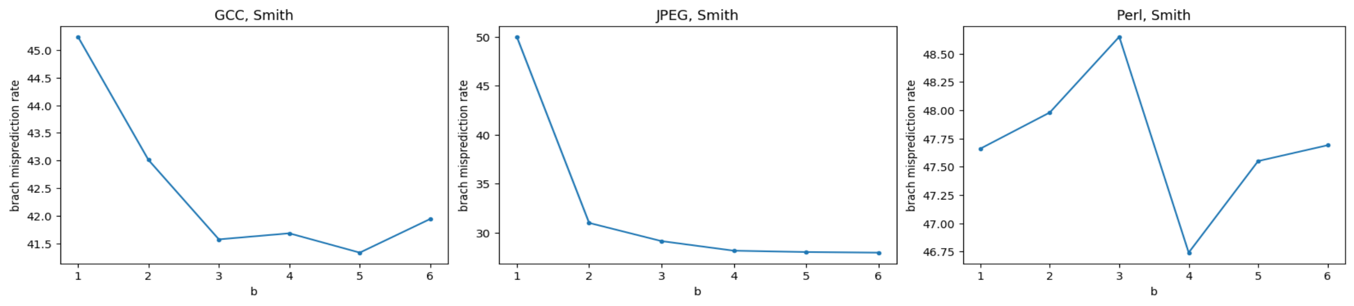

Fig. 4: Misprediction rate of Smith

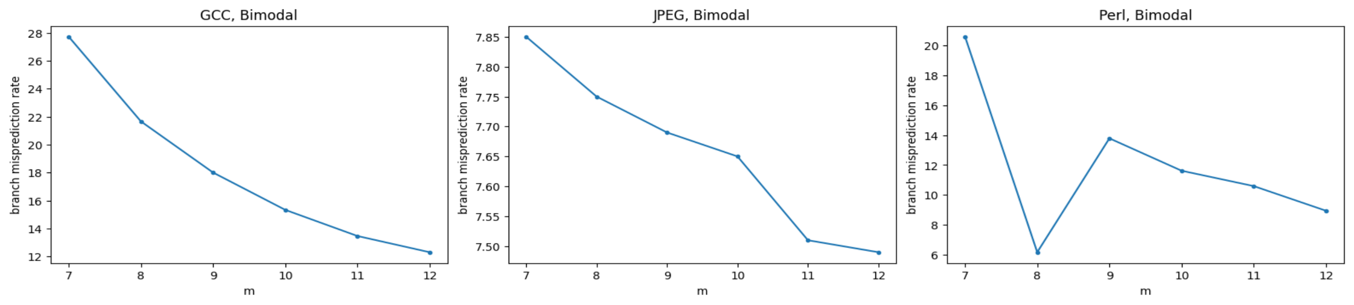

Fig. 5: Misprediction rate of Bimodal

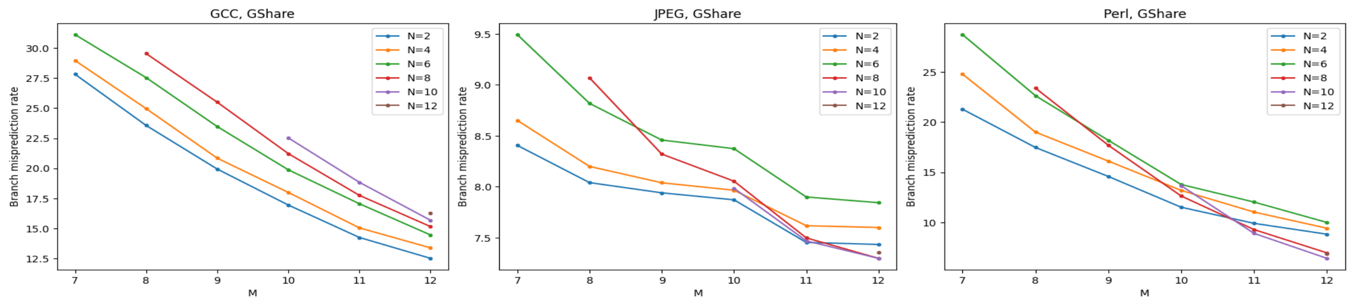

Fig. 6: Misprediction rate of Gshare

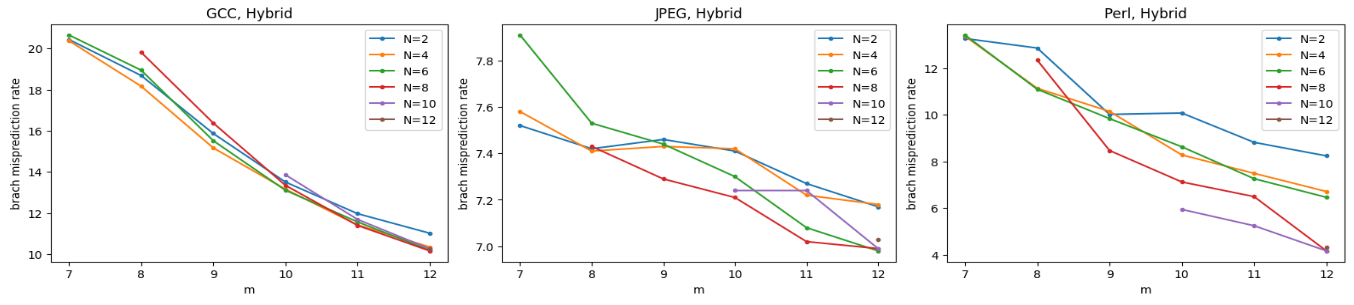

Fig. 7: Misprediction rate of Hybrid

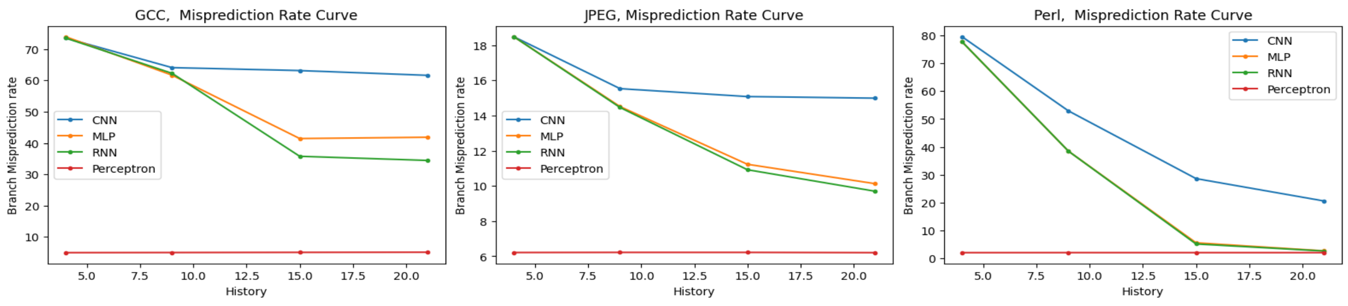

Fig. 8: Misprediction rate of neural networks

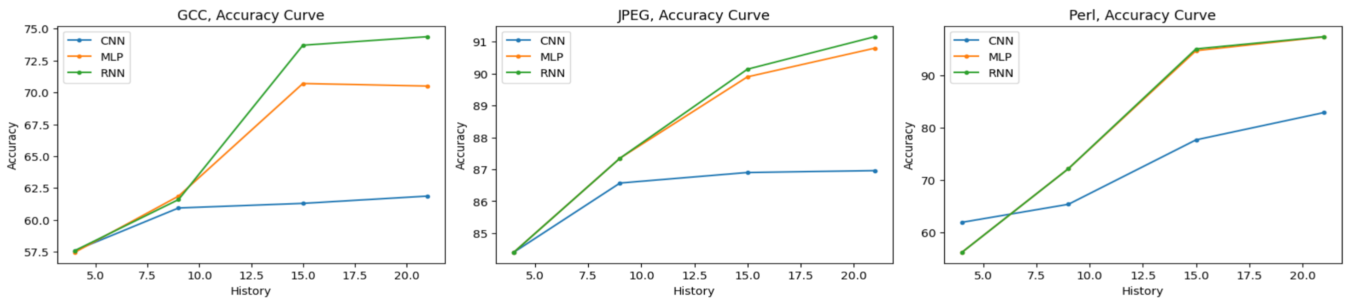

Fig. 9: Accuracy of neural networks

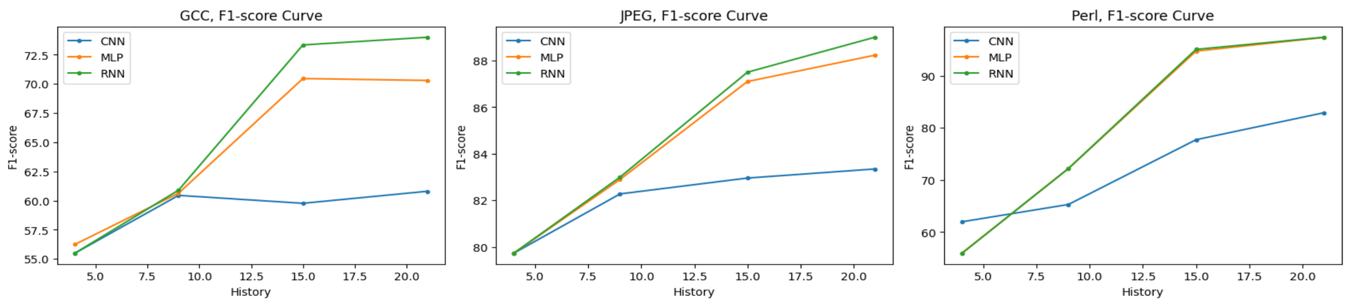

Fig. 10: F1 score of neural networks

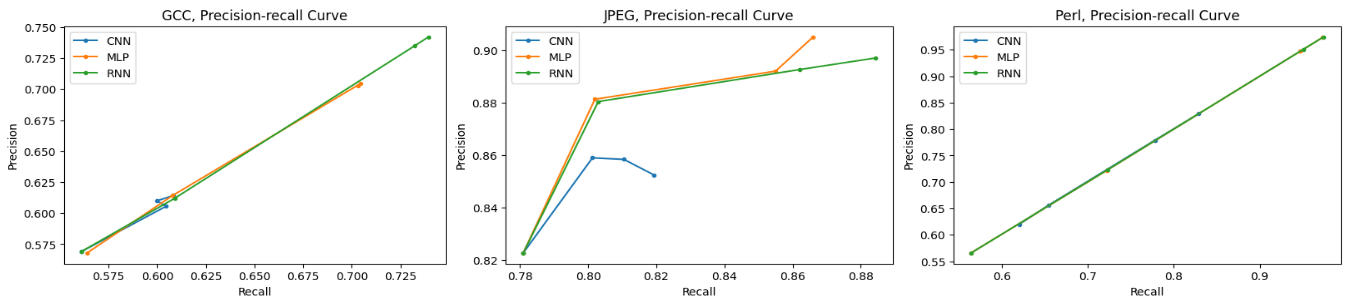

Fig. 11: Precision-recall curve of neural networks