An astute LVQ approach using neural network for the prediction of conditional branches in pipeline processor

Sweety Nain[1,*] and Prachi Chaudhary[2]

[1]Research Scholar, Department of E.C.E, D.C.R.U.S.T, Murthal, Haryana, 131001
[2]Assistant Professor, Department of E.C.E, D.C.R.U.S.T, Murthal, Haryana, 131001

Abstract

Nowadays, microprocessors use the deep pipeline to execute multiple instructions per cycle. The frequency and behavior of conditional instructions mainly affect the performance of instruction-level parallelism. However, recent processors still have problems with the correct prediction of conditional branches. Firstly, the perceptron neural network and global-based perceptron prediction has been exploited and implemented. Further, a new approach, linear vector quantization (LVQ) neural network, is explored and implemented to see its possibility and potentiality as a branch predictor in terms of accuracy rate. Simulation is performed by varying the parameter of hardware budget and the length of history register using different trace files for identification of the best branch predictor technique. The proposed LVQ perceptron branch predictor achieves an 85.56% accuracy rate using a hardware budget and an 86.36% accuracy rate in terms of history length by comparing the simulation results.

Keywords: branch prediction, perceptron branch predictor, pipeline, linear vector quantization, accuracy rate.

Received on 04 October 2020, accepted on 08 February 2021, published on 04 March 2021

doi: 10.4108/eai.4-3-2021.168865

[*]Corresponding author. Email: sweetynain28@gmail.com

1. Introduction

In a microprocessor design, the pipeline is a prime high-performance technology as it enables high clock rates and instruction-level parallelism (ILP). The recent generation of processors like Pentium has been towards deeper pipelines to allow the increased clock speeds. As the pipeline becomes more in-depth, the controlling hazard due to conditional branches incurs the instruction's execution flow. In this case, the pipeline would have to wait for the branch output before fetching the next instructions. Precise prediction of branches with high predictive accuracy is required to resolve the problem in the pipeline. The conditional branch to be identified in the pipeline during the real-time prediction process. During the fetch step in the pipeline, the branch result and the target address must be predicted when the branch is found. However, the accuracy of the correct prediction of conditional instructions can lead to better performance of the processor.

The branch prediction methods fall into two categories.

- Static Branch Prediction

This prediction is the simplest of all methods as it has a predetermined branch action during the entire process. In this, the prediction is fixed during the compile time.

- Dynamic Branch Prediction

In this prediction, the processor uses hardware to store information about recently executed branches and their outcomes. Mostly, dynamic branch prediction techniques are based on pattern history tables (PHT) of saturating counters. Saturating counter prediction is limited with the branch history register and local information of the recently executed branch.

However, there are many algorithms like bimodal prediction, index sharing prediction, global and local based prediction, and a hybrid-based prediction that is extensively implemented to predict conditional branches. These algorithms are straightforward and achieve a normal prediction accuracy range.

It is recently possible to use machine learning techniques to improve the performance of the processor by replacing the saturating counter with the number of the perceptron since neural networks are known to provide better prediction accuracy using artificial neurons. The use of artificial neurons in perceptron is relatively better as their training process is speedy.

The rest of the paper is as follows: Section 2 gives a literature survey of the different techniques used to predict branches. The machine learning-based branch predictors are proposed in section 3. Section 4 describes the simulation framework for the branch predictors. The simulation results of each branch predictors are presented in section 5. Finally, in section 6, the research paper concludes with future scope.

Contributions

In this paper, the concept of machine learning is explored and tested to predict conditional branches.

- Firstly, the perceptron based neural network and the global-based perceptron prediction has been exploited and implemented.
- Secondly, a novel approach linear vector quantization (LVQ) neural network is proposed and implemented to see their possibility and potentiality as a branch predictor in terms of accuracy rate.
- Simulation is performed by varying the parameter of hardware budget and the length of history register using different trace files for identification of the best branch predictor technique.

The rest of the paper is as follows: Section 2 gives a literature survey of the different techniques used to predict branches. The machine learning-based branch predictors are proposed in section 3. Section 4 describes the simulation framework for the branch predictors. The simulation results of each branch predictors are presented in section 5. Finally, in section 6, the research paper concludes with future scope.

2. Related Work

Many researchers have tried to compare various branch predictors to highlight the efficiency of their approaches. So, this section presents the state-of-the-art of different branch prediction strategies for the prediction of branches.

Calder [1] provided the prediction of conditional branches using static and compiler-time branch prediction. This prediction is based on the preliminary information about the program that the compiler can readily determine. The significant drawbacks of Calder prediction are unable to use the dynamical prediction of branches. However, his static compiler optimization scheme providing extra information to dynamic branch predictors.

Franklin et al.[2] used the local and global history for the identification of branches. This concept tracks the run time behavior of an instruction in the front-end of the pipeline. This mechanism has been established for each dynamic branch predictor that regulates the computation that affects the expected branch outcome.

Vinten and Florea [3] implemented the conditional branch uses a back-propagation algorithm as a multilayer perceptron. Based on the same prediction information, this approach predicts the target address for indirect jumps. Moreover, the author suggests this approach to be more efficient for the prediction of branches but increases the hardware cost and complexity.

Tarjan and Skadron [4] introduced the hashed concept using the combination of gshare and perceptron branch predictor. This proposed predictor reduces aliasing, having a low hardware budget, and increase the accuracy in correlating predictors.

Peram and Sudhakar [5] presented a piecewise neural branch predictor for improving the perceptron branch predictor's accuracy. In this predictor scheme, a hyperplane is utilized to choose the conditional branch prediction. The main feature of this predictor scheme is to remove the complexity and give rise to more accurate results for improving the processors' performance.

Smith [6] proposed a feed-forward network based on the concept of a machine learning method. Further, a combined predictor using a saturating counter is also analyzed to compare accuracy and mis-prediction rate. Unfortunately, this scheme enhances accuracy, but tradeoff occurs between the number of hidden units.

Mao et al. [7] proposed a deep learning-based algorithm for branch prediction. The author considers branch prediction in this paper as a classification problem and contrasts deep learning efficiency with current branch predictors.

Su et al. [8] proposed a correlation-based hybrid branch prediction for the conditional branch. This approach combines the concept of static as well as dynamic branch prediction. To significantly boost the branch prediction accuracy, the dynamic branch predictor uses the branch correlation data. At the same time, the static profile based correlation is used to identify the branches.

Shah and Prabhu [9] implemented a hybrid branch predictor with higher predictive capabilities than global branch predictors. In neighboring branches and local branches, the branch prediction accuracy is enhanced by basing prediction.

Jimenez [10] described two versions of perceptron predictors by taking the parameter of long history lengths. To explore the feasibility of predictors, a circuit-level design of the perceptron predictor is designed. Further, it shows that in modern CPUs, the complex perceptron predictor can be used by providing a simple CPU., quicker and more feasible than the hybrid predictor.

In literature, the branch predictors lack performance because of the utilization of smaller history length, lack of hardware budget, and counter-based system. For this, there is a lack of fast fetching and executing the process of instructions. The proposed method is intelligent, contributing to producing beneficial and accurate results by including mathematical based calculation and using the training module.

Table 1. Different branch prediction schemes used for branch prediction with features

Different branch predictors	Features and challenges	Reference
Smith Algorithm	Improve the performance by a small increment, but it does not use the store history tables of an instructions	[11], [12]
Two-level predictor	Uses two separate levels of branch history tables. However, the trade-off between sizes of two-tables occur	[13], [14]
Index sharing predictor	The size of the history table is large as compared to the two-level predictor's. Hashing together branch history register and PC leads to better accuracy in processor performance	[15], [16]
The agree Predictor	Reduce destructive aliasing interference by reinterpreting the pattern history table counter	[17], [18]
Hybrid branch predictor	Combine two or more predictor's to make one final prediction but sometimes partially misunderstand the hybrid path at the time of prediction	[31][21]
The piece-wise linear neural branch predictor	It provides much greater precision but dramatically increases the overhead of control pointing and recovery and the number of adders.	[22]–[24]

3. Machine Learning Branch Predictors

Recently, machine learning becomes a new research focus and significantly improves the performance of processors. In this section, the description of the machine learning-based branch predictors is explored with their algorithm of how they can be used to predict conditional branch instructions.

3.1. Perceptron based branch predictor

A perceptron is one of many processing elements within the artificial neural networks. A perceptron is a learning device that takes input values and combines them with weights to produce an output. Figure 1 presents the conceptual view of a perceptron model. Given a vector of inputs $x_i......x_n$ and a vector of weights $w_i......w_n$ the output of the perceptron is a dot product of data and weight. x_i is always set to 1 represent bias input. This allows the perceptron to learn its activation threshold. The output y is expressed in mathematical:

$$y = w_0 + \sum_{i=1}^{\infty}(x_i w_i) \quad (1)$$

Here,
y = Output of perceptron, w_0 = Bias weight, w_i = Perceptron weight, x_i Perceptron input
The output of the perceptron is a dot product of data and weight. The perceptron's output is y; if the $y \geq 0$ branch is predicted to be taken, else branch is expected to be not-taken.

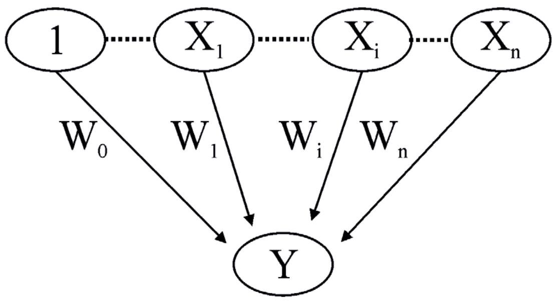


Figure 1. A simplest perceptron structure

The basic block diagram in figure 2 represents the role of perceptron for the prediction of branch instructions. For the prediction of conditional branch instructions, a perceptron uses a table in which n number of a perceptron is stored. A training algorithm is used to train the module when the outcome is not equal to the actual result.

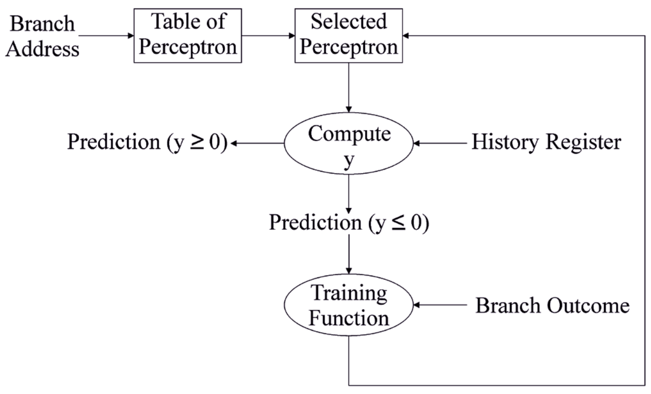


Figure 2. The conceptual model of the perceptron branch predictor

For the prediction of the branch using perceptron branch predictor, the following steps are to be taken:

(i) The branch address is hashed to the table of the perceptron.

(ii) Select the perceptron for computing the output.
(iii) Compute the branch prediction y
if $y \geq 0$, prediction result to be saved
if $y \leq 0$, prediction to be updated using train function
(iv) Train the selected perceptron using branch outcome.
(v) Update the trained perceptron back to the table.

Algorithm: Perceptron Based Branch Predictor

```
Step 1: Use the program counter to select the input branch;
Step 2: Get the weight vector of each input branch ;
Step 3: Compute output using weight vector and input branch;
Step 4: Make prediction based on output;
Step 4.1 If (prediction = incorrect or below threshold);
     then (adjust weight vector using train function);
Step 5: If (prediction = correct or above threshold);
     then {
      increment weight and return 1, if taken
      else
     decrement weight and return 0, if not-taken
     }
```

When the actual performance of the branch is known, a training algorithm is used to update the predictor. The training algorithm uses a threshold parameter to control the magnitude of the weight value. The threshold value is optimal to be $\theta = [1.93h + 14]$, where h represents the duration of the history bit. The following algorithm is used to train the value of the perceptron.

Algorithm: Train Perceptron function

```
if (y) $\neq t$ or ( |y| $\leq \theta$ ) then
for j = 0....h
do
{
$w_i = w_i + 1$, if ( $t = x_i$ )
or
$w_i = w_i - 1$, if ( $t \neq x_i$ )
}
end
```

With this algorithm, the perceptron trains its weight table, achieving more accuracy while predicting the branches. One of the limitations of using perceptron is they are only capable of learning linear separable function. The global perceptron branch predictor overcomes this limitation by using the linear inseparable function.

3.2. Global perceptron branch predictor

The global branch predictor is one of the best prediction schemes among the correlating branch prediction schemes. The prediction of this scheme is based on the history table of recently executed predicted branches. To index the bits in the history table, XOR to be used of the least significant bit of the currently executing branch address and the history of the recently completed branch instruction.

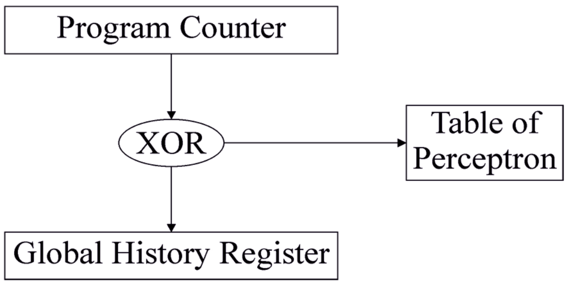


Figure 3. The global perceptron branch predictor fetches weights by indexing XOR of address

In this predictor, the perceptron table is indexed by the correlation of bits assigned by the XOR of branch address and the speculative global history register. The branch address holds the address of currently executing conditional instructions, whereas the global history register holds the instruction's prior information. The perceptron is trained according to their cumulative prediction. The great advantage of using this predictor for the branch prediction is that each weight is fetched with a different mapping to get a more accurate prediction.

Algorithm: Global Perceptron Branch Predictor

```
Step 1: Use the program counter to select the input branch;
Step 2: Get the weight vector of each input branch ;
Step 3: fetch the address in table of perceptron using xor function
Step 4: Make prediction based on output;
Step 4.1 If (prediction = incorrect or below threshold);
     then (adjust weight vector using train function);
Step 5: If (prediction = correct or above threshold);
     then {
      increment weight and return 1, if taken
      else
     decrement weight and return 0, if not-taken
     }
```

Table 2 depicts the example of XOR used in the global perceptron predictor. Using this indexing, they can predict some linearly inseparable branches and overcome the perceptron branch predictor problem.

Table 2. The bits XOR used in global perceptron predictor

Branch Address	Global History Register	XOR Indexing
0000 1010	0000 0001	0000 1011
0000 1110	1000 1010	1000 0100
1111 1001	0000 0001	1111 1000

1111 1110	1000 1111	0111 0001

The table utilization is significantly improved, which makes global predictors achieve higher prediction accuracy with the same hardware storage as compared to the perceptron branch predictor.

3.3. Proposed Linear Vector Quantization Neural Predictor

A proposed linear vector quantization (LVQ) predictor is based on the supervised competitive artificial neural network. This technique is associated with the neural network class of learning algorithms. LVQ consist of codebooks class of different parameter to refine the statistical analysis of any complex problem. In conditional branch instructions, the first codebook vector v_t represents the branch taken outcomes, and the second codebook vector v_{nt} represents the branch not-taken outcomes. For the correct predictions, the vector value is to be increased when the prediction to be accurate else the vector value is to be decreased.
For computing the output outcomes, the results are based on the hamming distance between the input vectors and the codebook vectors associated with that particular input. The hamming distance is calculated as :

$$HD(y) = \sum_{i=1}^{n} (x_i - v_i)^2 \qquad (2)$$

Here, y = Prediction Outcome , x_i = Input Vector , v_i = Codebook Vectors
To train the codebook vectors, the particular vector value is adjusted as :

- If the target value is equal to the prediction then update the codebook vector by
  $w_j(t+1) = w_j(t) - a(x - w_j(t))$ ..... (3)

- If the target value is not equal to the prediction then update the codebook vector by
  $w_j(t+1) \neq w_j(t) - a(x - w_j(t))$ .....(4)

Here, $w_j(t+1)$ = New Weight Value , $w_j(t)$ = Current Weight Value , a = Learning Rate, the range lies between $0 < a < 1$.

Algorithm: Linear Vector Quantization Branch Predictor

```
Function perceptron prediction lvq (pc,add: integer): boolen:
index = pc mod num
outcome  y =  bias [index] (initialize index and output value)
if (outcome == True):
$v_t$ = 1
for i in range(h)
if((self. ext & $v_t$) == 0):
x = -1
else
x = 1
outcome   y+= np.dot(.percept weight[index][i], x) (dot product calculation)
if( self.y >= 0 ): (making a prediction)
self.prediction = True
else
self.prediction = False
end if
end
```

The LVQ neural model has continuously trained the weights and provide faster processing than other neural models. This model's main advantage is to reduce the more massive data sets into a smaller number of codebook vector for the easy classification.

4. Simulation Framework

In this section, we describe the details of the simulator, trace-files, and parameters used to predict branches.

4.1. Simulator and trace file

Each branch predictor scheme's simulation is implemented on python numpy and pycharm simulator for visualization and computing the input trace files. The trace file is the text files with space-separated branch addresses and their actual outcomes. The trace files are trace1k, trace2k, trace5k, trace10k, trace20k, trace40k and trace files, including different instructions.

4.2. Influences of parameters

To evaluate each predictor scheme's (3) performance, the varying range of hardware budget and history length is taken. The hardware budget is related to the memory size of the processor in terms of kilobytes. (4) The hardware budget range is 4kb, 8kb, 16kb, 32kb, 64kb, and 80kb. At the same time, the history length relates to storing information on instructions in terms of bits. The range of history length is 4bit, 8bit, 16bit, 24bit, 28bit, and 32bit. The different size of trace files is used as input contains the branch address and the outcomes of the instructions. To identify the accuracy rate, each branch predictor is analyzed by varying the hardware budget and history length parameter using each trace file. The hardware budget-related to the memory size of the processor in terms of kilobytes. At the same time, the history length relates to storing information on instructions in terms of bits.

5. Experimental Results

This section quantifies the performance of branch predictors and compares the results in terms of their accuracy rate. For performance assessment, a set of trace files was used. To test the branch predictor's performance, we evaluated the impact of changing branch hardware budget and changing history length value with different track files. The statics results of accuracy rate is termed as :

$$Accuracyrate(\%) \equiv \frac{TotalNumberofAddressHit}{TotalNumberofInstructionExecuted}$$

Here,
Total number of address hit = Prediction on conditional branch instruction is equal to their actual address path.
Total number of instruction executed = Number of conditional instruction taken as an input.

5.1. Prediction based on hardware budget in term of accuracy rate

To evaluate the performance of branch predictors, different trace files are tested with varying budgets of hardware. The hardware budget related to the memory size of the processor in terms of kilobytes. The range of hardware budget is 4kb, 8kb, 16kb, 32kb, 64kb, and 80kb. Each predictor's accuracy is based on the actual prediction of the branch instruction accurately match with the predictor outcomes. The results of each branch predictor in term of accuracy rate is presented in table 4 to table 6. The comparison results show that the proposed LVQ branch predictor is more accurate and provides a higher accuracy rate when the hardware budget is varying.

Table 4. Accuracy Rate of Different Trace Files With Varying Hardware Budget of Perceptron Based Branch Predictor

	Hardware Budget (kilo-bytes)					
Trace File	4kb	8kb	16kb	32kb	64kb	80kb
1k	76.30%	76.00%	76.40%	76.50%	76.20%	76.10%
2k	83.70%	83.30%	83.35%	83.45%	83.35%	83.60%
5k	82.64%	82.06%	82.14%	82.36%	82.44%	82.49%
10k	76.12%	77.47%	82.33%	77.91%	80.75%	82.45%
20k	81.29%	82.08%	82.90%	82.60%	82.63%	82.56%
40k	77.78%	83.98%	85.05%	83.27%	85.18%	84.38%

Table 5. Accuracy Rate of Different Trace Files With Varying Hardware Budget of Global Perceptron Branch Predictor

	Hardware Budget (kilo-bytes)					
Trace File	4kb	8kb	16kb	32kb	64kb	80kb
1k	76.55%	71.50%	72.54%	71.40%	77.20%	74.30%
2k	80.60%	89.75%	78.10%	88.40%	77.30%	87.05%
5k	83.92%	72.72%	89.72%	71.34%	79.64%	89.78%
10k	87.23%	86.60%	86.48%	78.11%	86.10%	86.11%
20k	86.40%	77.24%	76.15%	77.33%	85.01%	74.98%
40k	86.12%	88.61%	88.06%	87.11%	77.83%	77.75%

Table 6. Accuracy Rate of Different Trace Files With Varying Hardware Budget of Proposed Linear Vector Quantization Branch Predictor

	Hardware Budget (kilo-bytes)					
Trace File	4kb	8kb	16kb	32kb	64kb	80kb
1k	79.25%	75.50%	77.54%	73.40%	79.20%	74.30%
2k	83.50%	91.35%	79.12%	89.30%	78.30%	85.05%
5k	86.88%	75.00%	90.62%	78.20%	80.56%	91.08%
10k	88.23%	89.60%	87.48%	79.10%	87.12%	88.10%
20k	87.42%	78.23%	76.25%	79.33%	86.10%	76.98%
40k	88.12%	90.01%	89.06%	89.10%	79.84%	79.75%

Table 7. Average Accuracy rate of each branch predictor by varying hardware budget

Hardware Budget	Perceptron Based Branch Predictor	Global Perceptron Branch Predictor	Proposed LVQ Neural Branch Predictor
4kb	79.63%	81.47%	85.12%
8kb	80.81%	82.07%	84.65%
16kb	82.02%	84.84%	86.57%
32kb	81.33%	85.94%	86.24%
64kb	81.75%	85.51%	85.57%
80kb	81.93%	81.66%	85.26%

Figure 4 shows how the accuracy rate is changed by increasing the hardware budget's size for the branch predictors. Prediction accuracy is the number of branches correctly predicted over the total number of branches. The size of the hardware budget increases with the amount of pattern history tables in which the information is processed.

The prediction accuracy is varied between 79.63% and 81.93% in perceptron based branch predictor. In the global perceptron branch predictor, the accuracy range varies between 81.47% to 85.94%. In the proposed LVQ branch predictor, the accuracy range varies between 85.12% to 86.57% and provides a better accuracy rate than the other two predictor schemes.

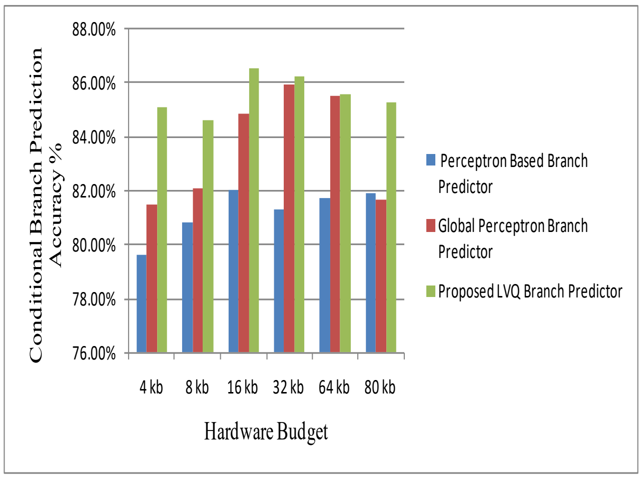


Figure 4. Average accuracy rate of different branch predictors by varying hardware budget

5.2. Prediction based on history length in term of accuracy rate

The impact of history length on the prediction accuracy has been studied for a while. So, to evaluate branch predictors' performance, different trace files are tested with varying history length. The range of history length is 4bits, 8bits, 16bits, 24bits, 28bits, and 32bits. Each predictor's accuracy is based on the actual prediction of the branch instruction accurately match with the predictor outcomes. The results of each branch predictor in term of accuracy rate is presented in table 8 to table 10.

Table 8. Accuracy Rate of Different Trace Files With Varying History Length of Perceptron Based Branch Predictor

	History Length (bits)					
Trace File	4bits	8bits	16bits	24bits	28bits	32bits
1k	76.80%	77.70%	77.50%	76.90%	75.60%	75.70%
2k	81.80%	82.75%	84.00%	83.80%	83.45%	83.80%
5k	84.42%	83.26%	83.52%	82.48%	82.36%	82.18%
10k	87.73%	87.28%	84.40%	80.70%	79.91%	80.31%
20k	83.52%	83.65%	82.91%	81.39%	82.60%	82.20%
40k	75.50%	82.75%	82.12%	77.90%	78.49%	81.00%

Table 9. Accuracy Rate of Different Trace Files With Varying History Length of Global Perceptron Branch Predictor

	History Length (bits)					
Trace File	4bits	8bits	16bits	24bits	28bits	32bits
1k	78.50%	78.90%	78.60%	79.50%	78.40%	79.50%
2k	83,75%	81.55%	79.20%	81.40%	78.42%	80.25%
5k	79.60%	83.84%	80.14%	82.84%	79.34%	81.50%
10k	83.50%	84.00%	82.90%	83.01%	82.75%	82.00%
20k	84.40%	84.45%	83.35%	84.40%	80.12%	81.36%
40k	76.60%	82.20%	82.80%	83.30%	81.20%	83.30%

Table 10. Accuracy Rate of Different Trace Files With Varying History Length of Proposed Linear Vector Quantization Branch Predictor

	History Length (bits)					
Trace File	4bits	8bits	16bits	24bits	28bits	32bits
1k	79.15%	78.10%	78.90%	80.01%	78.50%	79.60%
2k	8.60%	82.35%	80.10%	82.20%	79.51%	81.10%
5k	79.65%	83.35%	81.10%	83.29%	80.10%	82.20%
10k	84.41%	84.45%	83.91%	83.03%	83.35%	82.10%
20k	85.51%	85.56%	84.40%	84.42%	81.10%	82.20%
40k	78.90%	83.31%	83.31%	84.41%	82.21%	84.45%

Table 11. Average accuracy rate of each branch predictor by varying history length

History Length	Perceptron Based Branch Predictor	Global Perceptron Branch Predictor	Proposed LVQ Neural Branch Predictor
4 bits	82.85%	83.92%	84.04%
8 bits	82.92%	83.63%	83.90%
16 bits	82.46%	85.97%	87.04%
24 bits	81.05%	85.94%	86.14%
28 bits	80.96%	85.49%	88.57%
32 bits	80.83%	85.79%	88.51%

Figure 5 shows how the accuracy rate is changed by increasing the branch predictors' number of history lengths. The overall prediction accuracy is improved as the number of entries in the history length increase.

The prediction accuracy is varied between 80.83% to 82.92% in perceptron based branch predictor. In the global perceptron branch predictor, the accuracy range varies between 83.63% to 85.97%. In the proposed LVQ branch predictor, the accuracy range varies between 83.90% to 87.04% and provides a better accuracy rate than the other two predictor schemes.

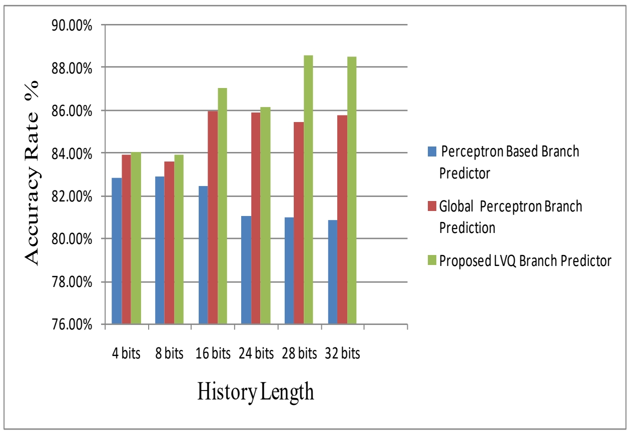


Figure 5. Accuracy rate of different branch predictors by varying history length

The proposed LVQ branch predictor improves the accuracy rate by varying both parameters (Hardware Budget and History Length) over the perceptron based branch predictor and global perceptron branch predictors.

5.3. Prediction Result in term of Confusion matrix and F-Score

To obtain the precision results, all the input trace files are profiled to determine each predictor scheme's branch decision. A confusion matrix for all the branch predictor scheme in each input trace files give some interesting insights of its actual behaviour. The confusion matrix for each branch predictor according to the input trace files has been represented in this way:

		Actual Value	
		Taken (T)	Not-Taken (NT)
Prediction Value	Taken (T)	TT	TNT
	Not-Taken (NT)	NTT	NTNT

Here,
Actual value means the actual outcome of the branch instruction.
Prediction value means the prediction uses during the run time on the branch instruction.
Taken means the prediction has been applied on the branch instruction.
Not-Taken means the prediction has not been applied on the branch instructions.
Further, the F-score is also calculated and it is a harmonic mean of precision and recall. The F-score is calculated using the given below formula:

$$F\_Score = (2 * \mathrm{Re}call * \mathrm{Pr}ecision) / (\mathrm{Re}call + \mathrm{Pr}ecision)$$

Here,
Recall describes how many of the actual taken values to be predicted correctly out of the model. It is useful when false-negative dominates false positives. The formula for calculating the recall is:

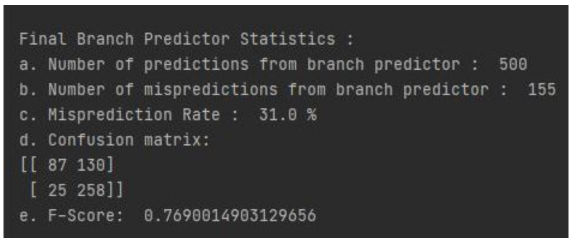
$$\mathrm{Re}call = (TT) / (TT + NTT)$$

Precision means the number of correct outputs given by the model out of all the model's correctly predicted positive values.

$$\mathrm{Pr}ecision = (TT) / (TT + TNT)$$

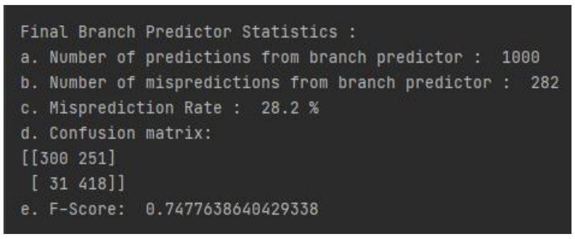
5.3.1 Confusion and F-score outcome of Perceptron Based Branch Predictor

The confusion matrix and the f-score of input trace file 1k and trace file 2k are presented in Figures 6 and 7. The result is changing according to the branch predictor and the input trace files. The similar results has been obtained using other trace files for perceptron based branch predictor.



```
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  500
b. Number of mispredictions from branch predictor :  155
c. Misprediction Rate :  31.0 %
d. Confusion matrix:
[[ 87 130]
 [ 25 258]]
e. F-Score:  0.7690014903129656
```

Figure 6. Results of input trace1k

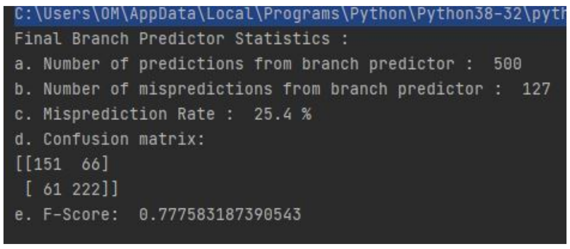


```
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  1000
b. Number of mispredictions from branch predictor :  282
c. Misprediction Rate :  28.2 %
d. Confusion matrix:
[[300 251]
 [ 31 418]]
e. F-Score:  0.7477638640429338
```

Figure 7. Results of input trace2k

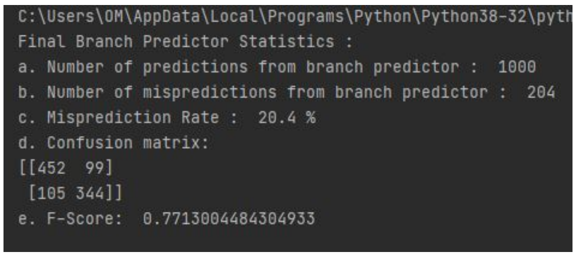
5.3.2 Confusion and F-score outcome of Global Perceptron Branch Predictor

The confusion matrix and the f-score of input trace file 1k and trace file 2k is presented in figure 8 and figure 9 respectively. The result is changing according to the branch predictor and the input trace trace files. This global perceptron branch predictor gives better results as compare with the perceptron based branch predictor. The result of f-score of trace file 1k and trace file 2k is improve by 0.01 and 0.02 respectively over the perceptron based branch predictor. The similar results has been obtained using other trace files for global perceptron branch predictor.



```
C:\Users\OM\AppData\Local\Programs\Python\Python38-32\pyth
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  500
b. Number of mispredictions from branch predictor :  127
c. Misprediction Rate :  25.4 %
d. Confusion matrix:
[[151  66]
 [ 61 222]]
e. F-Score:  0.777583187390543
```

Figure 8. Results of input trace1k

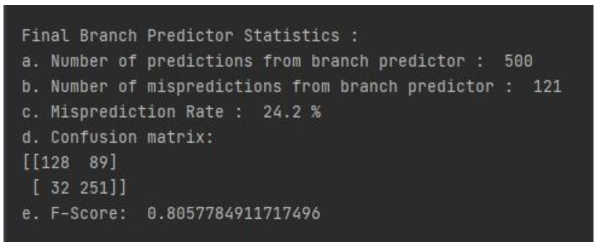


```
C:\Users\OM\AppData\Local\Programs\Python\Python38-32\pyth
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  1000
b. Number of mispredictions from branch predictor :  204
c. Misprediction Rate :  20.4 %
d. Confusion matrix:
[[452  99]
 [105 344]]
e. F-Score:  0.7713004484304933
```

Figure 9. Results of input trace2k

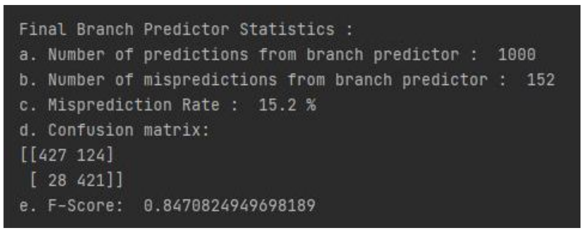
5.3.3 Confusion and F-score outcome of Proposed Linear Vector Quantization Branch Predictor

The confusion matrix and the f-score of input trace file 1k and trace file 2k is presented in figure 10 and figure 11 respectively. The result is changing according to the branch predictor and the input trace files. The similar results has been obtained using other trace files for global perceptron branch predictor. The proposed linear vector quantization branch predictor gives better results as compare with the perceptron based branch predictor and the global perceptron branch predictor. It improve the f-score by 0.04 and 0.10 using trace file 1k and trace file 2k respectively over the perceptron based branch predictor and 0.03 and 0.07 using trace file 1k and trace file 2k respectively over the proposed linear vector quantization branch predictor.



```
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  500
b. Number of mispredictions from branch predictor :  121
c. Misprediction Rate :  24.2 %
d. Confusion matrix:
[[128  89]
 [ 32 251]]
e. F-Score:  0.8057784911717496
```

Figure 10. Results of input trace1k



```
Final Branch Predictor Statistics :
a. Number of predictions from branch predictor :  1000
b. Number of mispredictions from branch predictor :  152
c. Misprediction Rate :  15.2 %
d. Confusion matrix:
[[427 124]
 [ 28 421]]
e. F-Score:  0.8470824949698189
```

Figure 11. Results of input trace2k

Table 12. The average accuracy rate of the branch predictors

Algorithm	Accuracy Rate (Hardware Budget)	Accuracy Rate (History Length)
Perceptron Based Branch Predictor	82.85%	83.92%
Global Perceptron Branch Predictor	82.92%	83.63%
Proposed LVQ Perceptron Branch Predictor	82.46%	85.97%

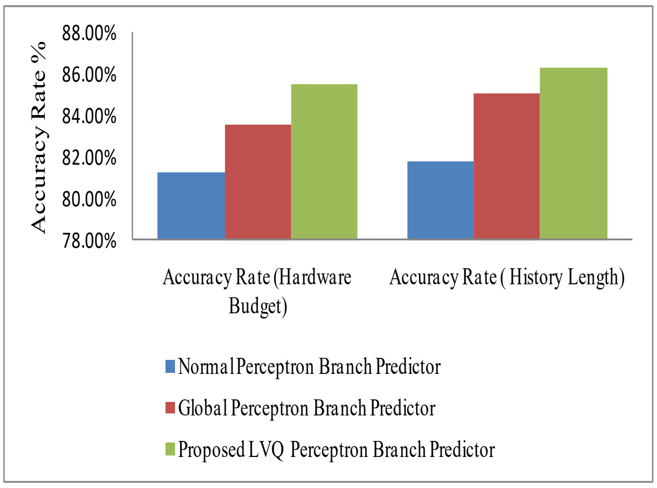


Figure 12. Average accuracy rate of the branch predictors

Table 12 shows the comparative analysis of different methodologies used for the prediction of the conditional branch. It clearly shows that:

- The accuracy rate of the proposed LVQ perceptron branch predictor is 4.32% higher than the perceptron based branch predictor and 1.98% higher than the global perceptron branch predictor as the effect of hardware budget is varying.
- Further, the results in varying-parameter history length also improve the accuracy rate of the proposed LVQ branch predictor by 3.28% higher than the perceptron based branch predictor, and 1.24% higher than the global perceptron branch predictor.

5. Conclusion and Future Trends

In this paper, the concept of artificial intelligence based neural networks is explored and tested to predict the conditional branches. Firstly, the perceptron based neural network and the global based perceptron prediction has been exploited and implemented. To add more preciseness, a novel approach LVQ neural network is proposed and implemented to see their possibility and potentiality as a branch predictor in terms of accuracy rate. This neural based branch predictors replace the saturating counters into the training function. Furthermore, the propose LVQ approach achieves better accuracy results than traditional branch predictors. Simulation is performed by varying the parameter of hardware budget and the history length using different trace files for identification of the best branch predictor.

The obtained results suggest that the proposed LVQ perceptron branch predictor provides increased accuracy rate of 85.56% by using a hardware budget and an

86.36% accuracy rate in terms of history length. The accuracy rate of the proposed LVQ perceptron branch predictor is 4.32% higher than the perceptron based branch predictor and 1.98% higher than the global perceptron branch predictor as the effect of hardware budget is varying. Further, the results in varying-parameter history length also improve the accuracy rate of the proposed LVQ branch predictor by 3.28% higher than the perceptron based branch predictor, and 1.24% higher than the global perceptron branch predictor. These improvements make this predictor a more promising choice for future processors.

According to this research paper, the concept of neural predictors could be a useful approach for understanding the process of branch predictors. Further, this concept can be used by using some other methods like back-propagation, support vector machine algorithm for better improvement in the accuracy rate.

Acknowledgements

I would like to thanks my guide Dr. Prachi Chaudhary for providing many helpful contributions during this paper.

References

[1] B. Calder, D. Grunwald, D. Lindsay, J. Martin, M. Mozer, and B. Zorn, “Corpus-based static branch prediction,” *ACM Sigplan Notices*, vol. 30, no. 6, pp. 79–92, 1995.

[2] R. Thomas, M. Franklin, C. Wilkerson, and J. Stark, “Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history,” *Proceedings of the 30th annual international symposium on Computer architecture*, vol. 31, no. 2, pp. 314–323, 2003, doi: 10.1145/859654.859655.

[3] L. N. Vintan and A. Florea, “A new branch prediction approach using neural networks,” *Proceedings of 10th International Symposium on Computers and Informatics SINTES*, no. 4, pp. 1–7, 2000.

[4] D. Tarjan and K. Skadron, “Merging path and gshare indexing in perceptron branch prediction,” *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 3, pp. 280–300, 2005, doi: 10.1145/1089008.1089011.

[5] S. Rao and P. K. Sudhakar, “An analysis to improve branch prediction accuracy by using neural branch prediction,” *International Journal for Modern Trends in Science and Technology*, vol. 3, no. 5, pp. 1–7, 2018.

[6] A. Smith, S. Diego, and L. Jolla, “Branch prediction with neural networks: Hidden layers and recurrent connections,” *Department of Computer Science University of California, San Diego La Jolla, CA*, vol. 9, no. 2, pp. 1–15, 2004, doi: 10.1.1.116.8548.

[7] Y. Mao, J. Shen, and X. Gui, “A study on deep belief net for branch prediction,” *IEEE Access*, vol. 6, pp. 10779–10786, 2017.

[8] X. Su, H. Wu, and Q. Yang, “An efficient wcet-aware hybrid global branch prediction approach,” in *Proceedings - 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016*, 2016, pp. 195–201, doi: 10.1109/RTCSA.2016.46.

[9] P. Z. Shah and S. U. Prabhu, “Hybrid learning-based branch predictor,” *International Journal of Engineering Research and Technology*, vol. 3, no. 8, pp. 1135–1139, 2014.

[10] D. A. Jimenez and C. Lin, “Neural methods for dynamic branch prediction,” *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 369–397, 2002, doi: 10.1145/571637.571639.

[11] C. H. Perleberg and A. J. Smith, “Branch target buffer design and optimization,” *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 396–412, 1993, doi: 10.1109/12.214687.

[12] J. E. Smith, “A study of branch prediction strategies,” in *Proceedings of the 8th annual symposium on Computer Architecture*, 1981, pp. 135–148, doi: 10.1145/285930.285980.

[13] C. Egan, G. Steven, P. Quick, R. Anguera, F. Steven, and L. Vintan, “Two-level branch prediction using neural networks,” *Journal of Systems Architecture*, vol. 49, no. 12–15, pp. 557–570, 2003, doi: 10.1016/S1383-7621(03)00095-X.

[14] J. Goyal, Shivam and Singh, “Two-level alloyed branch predictor based on genetic algorithm for deep pipelining processors,” *International Journal of Modern Education and Computer Science*, vol. 9, no. 5, pp. 27–33, 2017.

[15] J. Lu *et al.*, “The performance of runtime data cache prefetching in a dynamic optimization system,” *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, pp. 180–190, 2003, doi: 10.1109/MICRO.2003.1253194.

[16] A. Pandey, “Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor,” in *International Conference on Inventive Computation Technologies (ICICT)*, 2016, vol. 2, pp. 1–4.

[17] R. Parihar, “Branch Prediction Techniques and Optimizations,” *University of Rochester, NY, USA*, 2015.

[18] S. Mittal, “A Survey of Techniques for Dynamic Branch Prediction,” pp. 1–37, 2018, doi: 10.1145/2893356.

[19] S. Otiv, K. Garikipati, M. Patnaik, and V. Kamakoti, “H-Pattern : A hybrid pattern based dynamic branch predictor with performance based adaptation,” in *Proc. 4th JILP Workshop Comput. Architecture Competitions: Championship Branch*

Prediction, 2014, pp. 2–5.

[20] M. Evers, P.-Y. Chang, and Y. N. Patt, "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches," *Proceedings of the 23rd annual international symposium on Computer architecture - ISCA 96*, pp. 3–11, 1996, doi: 10.1145/232973.232975.

[21] J. Karimpour, S. Lotfi, and A. Tajari Siahmarzkooh, "Intrusion detection in network flows based on an optimized clustering criterion," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 25, no. 3, pp. 1963–1975, 2017, doi: 10.3906/elk-1601-105.

[22] D. A. Jiménez, "An optimized scaled neural branch predictor," *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 113–118, 2011, doi: 10.1109/ICCD.2011.6081385.

[23] P. Eng, L. N. Vin, U. Lucian, and F. De Inginerie, "Dynamic neural branch prediction fundamentals," 2016.

[24] M. Palermo, "The combined perceptron branch predictor," in *European Conference on Parallel Processing, springer*, 2005, pp. 487–496.