SN Computer Science (2021) 2:360
https://doi.org/10.1007/s42979-021-00748-9

ORIGINAL RESEARCH

Efficient Hardware Implementation of Decision Tree Training Accelerator

Rituparna Choudhury[1] · **Shaik Rafi Ahamed**[1] · **Prithwijit Guha**[1]

Received: 15 April 2021 / Accepted: 12 June 2021 / Published online: 26 June 2021
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2021

Abstract

In this paper, a serial architecture for acceleration and implementation of Decision Tree (DT) training algorithm has been proposed. This architecture is compatible with 32-bit integer as well as fixed-point training data. In the worst case scenario, the FPGA implementation of the proposed architecture for Two Means DT (TMDT) algorithm is proved to run at least 28× faster than conventional C4.5 training algorithm widely used in many machine learning classifications. The proposed architecture is implemented on FPGA platform operating at maximum frequency of 62 MHz. Further, the hardware implementation is proved to run at least 10× faster than the software implementation in worst condition. This design has been tested on five binary datasets of variable size and dimension. Thus, the proposed hardware realisation is compatible to wide range of training datasets.

Keywords Machine learning · Decision Tree · FPGA · Serial architecture · Training accelerator

Introduction

Decision Tree (DT) algorithms are widely used for classification and regression in many Machine Learning (ML) applications. Due to simplicity, DT is more suited to low power and low cost applications as compared to more complex algorithms such as neural networks (NNs) or support vector machines (SVMs) [1]. The basic structure of this algorithm comprises of tree with split or decision nodes and leaf or label nodes. It partitions data in top–down approach. The data first enters through the root node and are continuously classified by decision nodes and sent to appropriate child node till it arrives at a leaf node where it is assigned a label. This algorithm has two phases, training and inference. In the training phase, the tree parameters are tuned according to the data space [14]. During training, growth of the tree is controlled by termination condition. Then pruning is applied to the full-grown tree to avoid over-fitting or memorisation of data. Over-fitting results in poor generalisation and thus affects tree performance for unseen data. Several DT training algorithms have been reported in the literature [6, 9, 10, 12, 19]. Quadratic-neuron-tree (QUANT) proposed by in [19], implemented a quadratic neuron in the each node of the tree at the cost of increased complexity. The adaptive high-order neural tree (AHNT) [9] could classify a large set of multi-dimensional data. Each node in the tree either had higher order perceptrons (HOPs) which divided the data space arbitrarily or had first order nodes which used hyper-planes to divide the input space which produced very large trees for complex data-sets. Davey et al. proposed a stochastic competitive evolutionary neural tree in [6], which could hierarchically classify unlabelled multi-dimensional data and the number and structure of competitive nodes could be self determined by the tree. But it had an inferior performance for reduced dimension data-sets. A self-organizing neural tree has been discussed in [12], where growth processes are applied to construct the tree. In these process, the structure and performance is optimised based on the classification efficiency of one or several nodes. The

This article is part of the topical collection "Hardware for AI, Machine Learning and Emerging Electronic Systems" guest edited by Himanshu Thapliyal, Saraju Mohanty and VS Kanchana Bhaaskaran.

✉ Rituparna Choudhury
ritup176102101@iitg.ac.in

Shaik Rafi Ahamed
rafiahamed@iitg.ac.in

Prithwijit Guha
pguha@iitg.ac.in

[1] Indian Institute of Technology Guwahati, Guwahati, India

structure-parameter-adaptive (SPA) tree proposed in [10] is able to adapt to a changing environment both structurally and parametrically. The SPA tree growth is enhanced by new concepts and the tree shrinks at disposition of old concepts. Song et al. in [18] proposed a Structurally Adaptive Intelligent Neural Tree (SAINT). This tree performs hierarchical partition of input pattern space using tree structured networks which consist of sub-networks with topology preserving mapping ability. This tree is able to automatically detect the size and structure suitable for classification of large set of data. These algorithms are highly complex and implemented in software which results in larger run-time.

So, there has been a shift towards hardware implementation of classification algorithms in recent years [13]. It has increased the speed and thus resulted in a much faster training [15]. The DT training consumes greater amount of time as compared to classification. So, speeding up the training phase is very much important for efficient implementation of DT algorithms. Moreover, the software executes the code sequentially which leads to longer training time. Hardware implementation is a good solution to speed up the execution due to provision of higher parallelism which reduces the execution time. Furthermore, training on hardware would offer possibilities of on-device training and reduce the security risk of data leak which happens in case of training on cloud.

Some classification architectures for C4.5 and their implementation on FPGA are presented in [3, 11, 16, 17, 20, 22]. The architecture proposed in [3] is used to filter the sensor data. Here a serial architecture implemented on both FPGA and Von-Neumann CPU is compared and FPGA architecture is found to be less power hungry as compared to software counterpart but at the cost of decrease in throughput. Saqib et al. in [16] proposed a pipelined architecture to achieve faster classification. Tong et al. proposed one architecture which provided balanced classification in shorter time and another architecture produced an optimised tree with less area in [20]. The resource constrained architecture for seizure detection proposed by Shoaran *et al.* in [17] reduced the resource consumption and thus consumed very less power. Thus FPGA is a more convenient implementation platform due to its low power and re-configurability feature. These features makes them ideal for implementing training of classification algorithm. Moreover, FPGA can be easily re-trained to suit new data while improving classification performance at lower cost. The hardware architecture for texture sea-state classification proposed in [11] performs automatic texture recognition of sea-states. It allows to select an appropriate algorithm for target detection. The hardware proposed by Yang et al. in [22] proposes a spike classification SOC to reduce the data rate of a brain-machine interface. The classification was realised using a decision tree based classification which acquired an accuracy comparable to methods based on L1 distance.

The design proposed in [4] implemented k-means algorithm classification parallely on multiple FPGA to achieve 10× speed-up as compared to software implementation. The Euclidean distance calculation in k-means algorithm was replaced by Manhattan and Max distance in [8] which enabled the implementation without multipliers. The k-means tree implementation proposed in [21] realises a kd-tree pruning on the search space. This pruning leads to almost 5× lesser computation as compared to conventional k-means classification. The FPGA implementation of Classification and Regression Tree (CART) training algorithm on Convey HC-1 server proposes a parallel and pipelined architectural design[5]. A speed-up by a factor of 2 is achieved by running five FPGAs in parallel to process the data. However, this method requires more resources. To achieve better speed-up without increasing resource consumption, in this paper, we proposed a serial 32-bit architecture and its FPGA-based implementation or training of TMDT algorithm. This algorithm is similar to k-means algorithm. The rest of the paper is arranged as follows. "Proposed TMDT Algorithm" elaborates the TMDT algorithm. "Proposed Hardware Architecture" describes the proposed design. "Results and Discussion" presents the results and discussion. "Conclusion and Future Work" presents the future work and conclusion.

Proposed TMDT Algorithm

This work proposes a training hardware for binary classification using TMDT algorithm. This is derived from a competitive tree algorithm proposed in [2]. The TMDT hierarchically partitions the input space in a top–down manner as shown in Fig. 1. These partitions are achieved by decision functions hosted by split nodes of the tree. The tree terminates at leaf nodes. Each split node hosts two means.

Algorithm 1: TMDT Algorithm

```
while NodeIndex ≤ MaxNodeNum do
    Load node data;
    Calculate initial mean;
    while DataIndex ≤ NodeData do
        Calculate distance(μ^k_ql, x^k) & distance(μ^k_qr, x^k);
        if distance(μ^k_ql, x^k) ≤ distance(μ^k_qr, x^k) then
            Update μ^k_ql;
        else
            Update μ^k_qr;
        end
    end
end
while DataIndex ≤ NodeData do
    Calculate distance(μ_ql, x^k) & distance(μ_qr, x^k);
    if distance(μ_ql, x^k) ≤ distance(μ_qr, x^k) then
        Update left child node;
    else
        Update right child node;
    end
end
while NodeIndex ≤ MaxNodeNum do
    if (purity(Node) ≥ θ_p) or (NodeData ≤ ζ) then
        Set current node as label node;
        Set label as max(class1,class2);
    else
        Preserve the node structure;
    end
end
```

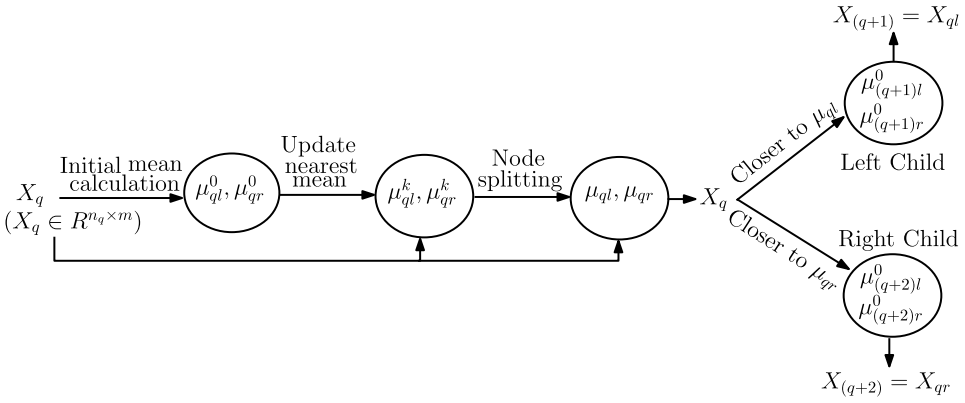

Fig. 1 Illustrating the flow of TMDT. The split node q is initialised with two mean vectors μ^0_{ql} and μ^0_{qr}. $\mathbf{X_q}$ first updates the two mean vectors then it is routed by these two mean vectors. The input is routed to the left child if it is closer to μ_{ql} and to right, otherwise. Thus $\mathbf{X_q}$ is split into two sets $\mathbf{X_{q+1}}$ and $\mathbf{X_{q+2}}$ which are assigned to the two child nodes of q , i.e., (q+1) and (q+2)

The decision function at each split node uses these means to route the node inputs to appropriate children nodes.

Let $\mathbf{X}_q = \{(\mathbf{x}_q^{(i)}, y_q^{(i)}); i = 1, \ldots n_q\}, (\mathbf{x} \in \mathcal{R}^m, y \in \{0, 1\})$ be the input to the split node q. This training dataset contains n_q data instances with labels ($y = 0$ or $y = 1$). The two means at the split node are initialized as mean vectors $\mu_{ql}^{(0)} = \mu_q^0$ and $\mu_{qr}^{(0)} = \mu_q^1$. These initial mean vectors μ_q^0 and μ_q^1 are calculated as average of data instances with label 0 and 1 respectively. In the second stage, the instances of $\mathbf{X}_q$ are again passed through the node to update the mean vectors. The training data $\mathbf{x}_q^{(k)}$ $(k = 1, \ldots n_q)$ updates its closest mean vector between $\mu_{ql}^{(k)}$ and $\mu_{qr}^{(k)}$ using an update rate $\beta \in (0, 1)$ (Equation 1). The value of β is empirically chosen as 0.8 for maximizing performance over different datasets.

$$\mu_q^{(k)} = (1 - \beta)\mu_q^{(k-1)} + \beta \mathbf{x}^k \tag{1}$$

The mean update stage is followed by the splitting of node input dataset $\mathbf{X}_q$. The dataset $\mathbf{X}_q$ is split into left and right children datasets $\mathbf{X}_{ql}$ and $\mathbf{X}_{qr}$ $([\mathbf{X}_{ql} \cup \mathbf{X}_{qr} = \mathbf{X}_q] \wedge [\mathbf{X}_{ql} \cap \mathbf{X}_{qr} = \varphi])$ for further processing by the respective left (q_L) and right (q_R) child node of q. The splitting is performed using the following decision function.

$$\mathbf{x}_q^{(k)} \in \begin{cases} \mathbf{X}_{ql} & \| \mathbf{x}_q^{(k)} - \mu_{ql} \|_2 < \| \mathbf{x}_q^{(k)} - \mu_{qr} \|_2 \\ \mathbf{X}_{qr} & \| \mathbf{x}_q^{(k)} - \mu_{ql} \|_2 > \| \mathbf{x}_q^{(k)} - \mu_{qr} \|_2 \end{cases} \tag{2}$$

An input data $\mathbf{x}_q^{(k)} \in \mathbf{X}_q$ is routed to $\mathbf{X}_{ql}$ (or $\mathbf{X}_{qr}$) if it is closer to μ_{ql} (μ_{qr}). This process of dataset splitting and split node mean update starts from the root node and is recursively continued until the termination at the leaf nodes. A fully grown binary decision tree (two children of split nodes) of depth d (root node is considered to be at zero depth) has 2^d leaf nodes and $2^d - 1$ split nodes. This work allows the decision tree to grow to its full depth (decided by user). Afterwards, pruning conditions are applied for prevention of over-fitting of tree.

This work uses two post pruning conditions for removing nodes from a tree. First, pruning due to insufficient input data to node. Here, the node q is not subjected to further splitting if its input dataset size $| \mathbf{X}_q | = n_q$ is lesser than a certain threshold ζ. The threshold ζ signifies the sufficient number of data instances required to perform a split. The second pruning condition uses a node purity criterion. The decision tree hierarchically partitions the input space into different regions. The leaf nodes correspond to such partitions. Ideally, these regions must contain data instances from either class $y = 1$ or $y = 0$. Practically, these partitions contain a mixture of data instances coming from both classes. However, in most cases, these regions have a dominant category. Let, p_q^0 and p_q^1 be the respective fractions of data instances from categories $y = 0$ and $y = 1$ respectively in node q. The node q is declared a leaf node if $max(p_q^0, p_q^1) > \theta_p$. Here, θ_p is a purity threshold. For example, $\theta_p = 0.95$ signifies that the dominant category should have more than 95% instances to form a leaf node. Each leaf node is tagged by the class label (y) of its dominant category.

An input data instance $\mathbf{x}$ traverses a learned two means decision tree to reach the leaf nodes through split nodes. Each split node q evaluates the distance of input $\mathbf{x}$ from μ_{ql} or μ_{qr} and accordingly routes it to the next left or right child node. The input traverses a particular branch of the tree and reaches a terminal leaf node hosting a category label. Finally, $\mathbf{x}$ is classified with the category label of the leaf node.

The complexity of TMDT algorithm is much less as compared to C4.5 Decision Tree. The C4.5 algorithm requires $n_q \times m \times \log n_q$ number of sort operations in each node to determine the split criteria for $n_q \times m$ dimension dataset for a node q. In contrast, TMDT requires only $4 \times n_q$ compare (2 compare operations each for update and split phase) and n_q update operations only for the same dataset. Thus the TMDT runs much faster than the C4.5 algorithm for the same dataset.

Proposed Hardware Architecture

Overall Architecture

In this section, we developed a novel 32-bit hardware architecture as shown in Fig. 2. In order to implement the proposed TMDT algorithm discussed in "Proposed TMDT Algorithm", this architecture is designed in serial fashion to optimise resource consumption and minimise power. In this architecture, the first block is mean initialisation module where, for each iteration, the data is loaded from the data memory and added to $\boldsymbol{\mu_l}$ or $\boldsymbol{\mu_r}$ obtained from previous iteration and result is forwarded to node memory after processing complete data stored in data memory. If the data label is 0, then it is added to left mean $\boldsymbol{\mu_l}$ otherwise, to $\boldsymbol{\mu_r}$. The data number counter is compared to total node data. When data number equals the total node data, then the final value of $\boldsymbol{\mu_l}$ and $\boldsymbol{\mu_r}$ is obtained by dividing $\boldsymbol{\mu_l}$ and $\boldsymbol{\mu_r}$ by total number of data in that node belonging to class 0 and 1 respectively to obtain the average. This division operation is executed using shifters to speed up the division. To divide a number X by 2^G, X is shifted by G bits which do not incur any accuracy loss as $2^G << X$. If the divisor cannot be decomposed into power of 2, then it is approximated to nearest value of 2^G. In this way, two means for each node is calculated and stored as initial means in the node memory which constitutes the MEAN INITIALISATION MODULE. This module consists of a multiplexer to select the nearest mean. The select line of the multiplexer is connected to output of comparator. The

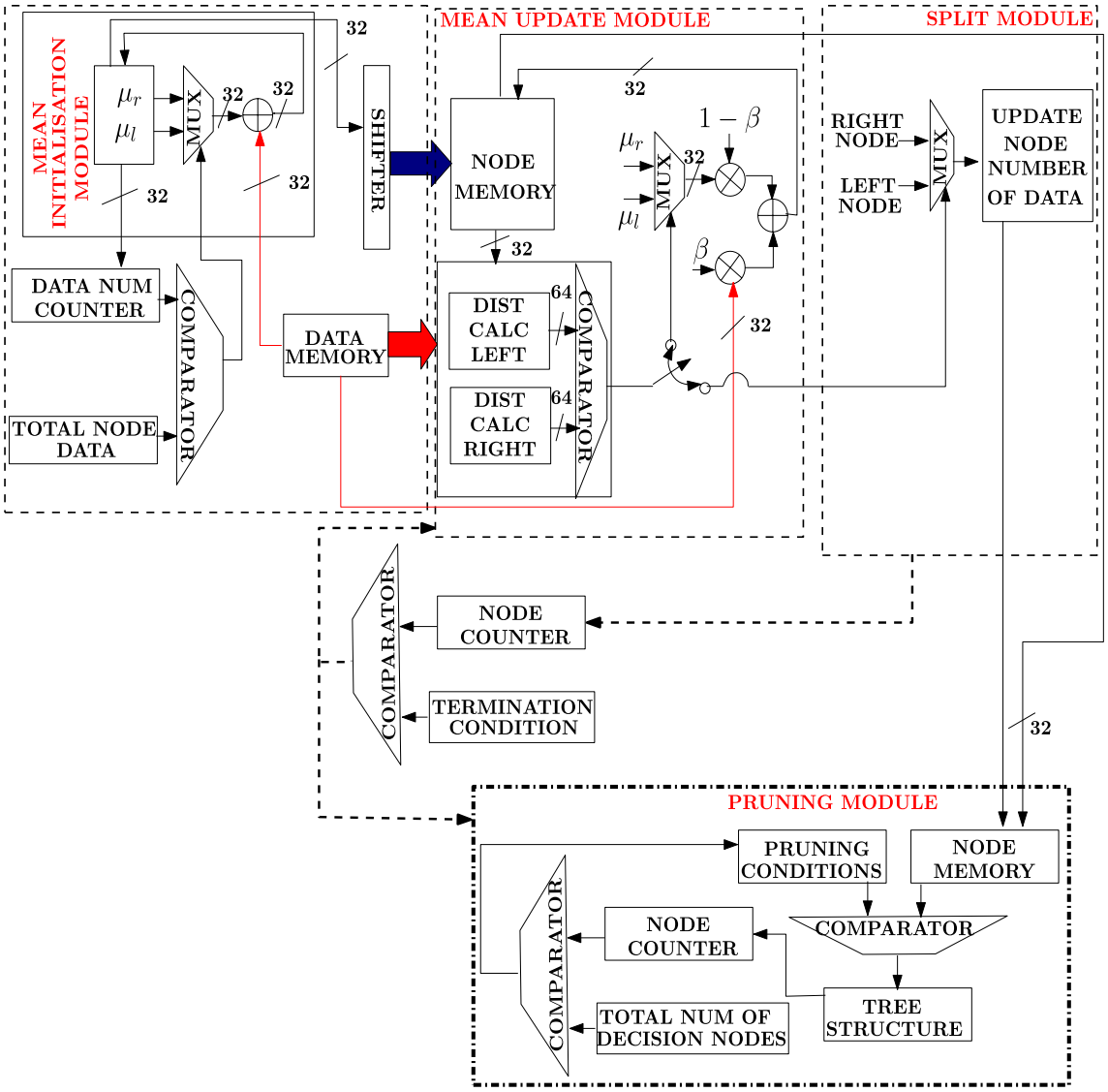

Fig. 2 Proposed Hardware Architecture for TMDT Algorithm

input to this comparator is TOTAL NODE DATA and current data number under consideration (DATA NUM COUNTER). The output of the comparator activates the multiplexer if current data number is less than TOTAL NODE DATA otherwise the final mean is calculated and stored in NODE MEMORY. In the subsequent phase, mean update and split module is executed for the node. In the MEAN UPDATE MODULE, either $\boldsymbol{\mu_l}$ or $\boldsymbol{\mu_r}$ is updated by the data-instance. The mean closer to data, depending on the distance calculated by distance calculator left and distance calculator right, parallely, is updated according to mean update equation (1). The details of distance calculator architecture is presented in next sub-section. The multiplexer (MUX) selects the mean closer to the data. The select line of this MUX is connected to the output of comparator comparing left and right mean distance with data-instance (DIST CALC LEFT and RIGHT respectively). Once the mean update is done then the updated mean is stored back to the NODE MEMORY. Once all the data belonging to that node is passed for mean update, then the same distance calculators are re-used to calculate the distance between the updated mean stored in node memory and the node data in the split module using a switch. This switch is implemented using a multiplexer whose control signal is connected to completion signal of splitting module. Once splitting is complete then the switch is connected to update module. This minimises resource utilisation as distance calculator uses a huge number of multiply and addition operations.

In the SPLIT MODULE, depending on whether the data is closer to $\boldsymbol{\mu_l}$ or $\boldsymbol{\mu_r}$, the data is either sent to the left child

or right child node selected by the multiplexer. The select line of this multiplexer is connected to the same distance comparator used in MEAN UPDATE MODULE through a switch. Then the node number of data and node memory is updated accordingly with the corresponding node number. Once the mean update and splitting is done, then the NODE MEMORY and DATA MEMORY is updated for that node and the initial means of it's child nodes are calculated in the MEAN INITIALISATION module. The node counter is then incremented and compared with the termination condition. If the termination condition is satisfied, then the PRUNING MODULE starts execution. Otherwise, next node is loaded in the mean update module. Then, SPLIT MODULE and MEAN INITIALISATION MODULE is executed for the node and its child nodes respectively. This process is repeated for all the nodes till the maximum depth is reached. In the pruning module, discussed later, the two post-pruning conditions as discussed in "Proposed TMDT Algorithm" are tested and nodes which satisfies the pruning conditions are set as leaf nodes and their children nodes (if any) are pruned or removed from NODE MEMORY.

Distance Calculator

The parallel hardware architecture for distance calculator module is shown in Fig. 3. The distance calculator is parallelised to the subtraction and squaring used in euclidean distance calculation simultaneously. Then all squared components are added to compute final distance using minimum number of clock cycles.

Pruning Module

As shown in Fig. 4, in pruning module, two comparators are used parallely to compare the purity of corresponding node data with purity threshold θ_p and the total node data n_q with data threshold ζ. The output of comparator is then fed into the OR gate. The output of OR gate is connected to multiplexer which sets the node as either split node or leaf node in tree memory and accordingly updates the children node.

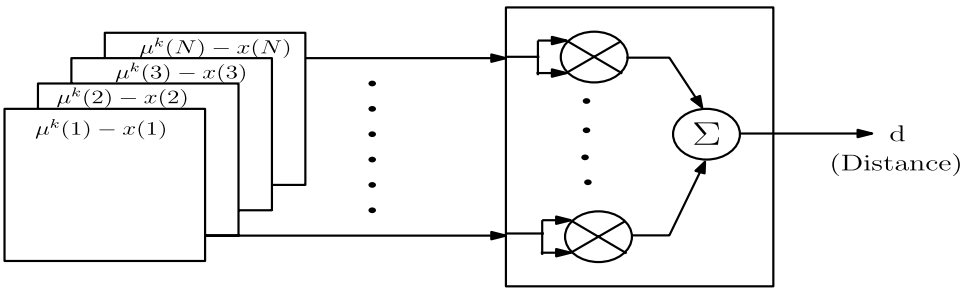

Fig. 3 Hardware architecture for distance calculator

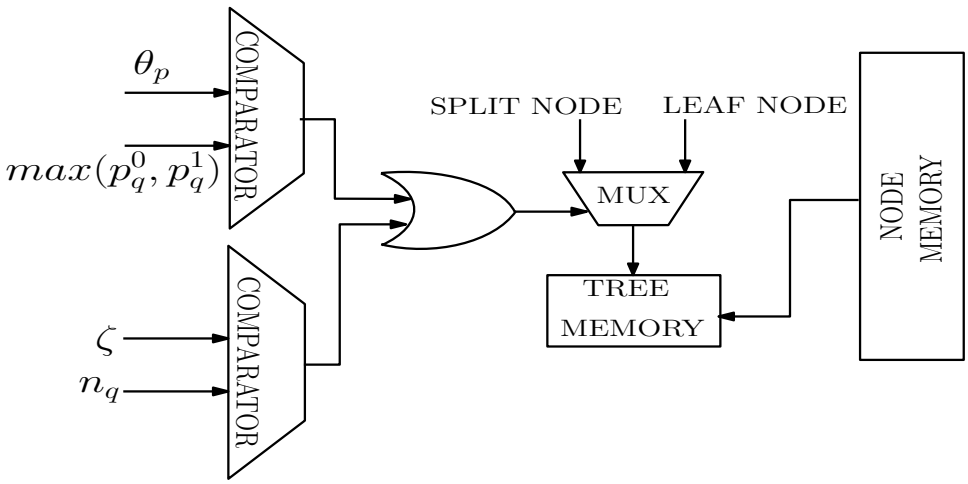

Fig. 4 Hardware architecture for pruning module

After all the nodes are processed and the termination condition is satisfied then the pruning module is executed to check the split condition for decision nodes. The decision nodes which do not satisfy the split condition are labelled as leaf node and their child nodes are deleted from node memory. Then the tree structure is updated and stored in the memory. Once this split condition check is completed for all decision nodes, then the training is complete and hardware is ready for classification.

Results and Discussion

This hardware was implemented on xcvu13p-flga2577-3-e Virtex Ultrascale+ FPGA board which uses 16nm technology. The design was implemented using Verilog HDL in Vivado 2017 and no high-level synthesis tool was used. All hardware results are post-implementation results. The maximum operating frequency of this design is 62 MHz. This hardware is able to support 32-bit fixed-point data. The software implementation was done on Intel core i5 processor running at 3.20 GHz. The design was tested for five balanced binary datasets, viz., skin, occupancy, activity recognition 1, activity recognition 2 and mammography [7]. The skin dataset has 38*k* data points each having four attributes, i.e., R, G, B pixel and binary value corresponding to skin or non-skin data. The occupancy has 10.124*k* data points each having six attributes, CO2, relative humidity, temperature, light, humidity ratio and whether the room is occupied. The activity recognition1 and activity recognition2 datasets has 14.39*k* data points each and distinguishes between whether a person is lying or sitting and whether a person is walking or standing respectively. The classification is done depending on six attributes, viz., five mean of reading from sensors placed in chest and ankles of subject and a binary number indicating whether the person is lying or sitting in case of activity recognition 1 and a binary number indicating walking or standing in case of activity recognition 2. The mammography dataset consists of 793 data points each having six attributes, size, shape, margin, density, severity and whether the tumor is benign or malignant.

In this design, the max depth d was set as 3 (considering node 0 at depth 0). For these datasets, after experimentation, the impurity threshold θ_p was set as 0.95 and the minimum number of data points for a node to be split node, i.e., ζ was set as 20. The update factor β was fixed at 0.8 for this architecture. The distribution of data for Skin dataset attribute or dimension 0 and 1 from parent node to child node till depth 2 is shown in Fig. 5. As shown in the graph, the separability of data for class 0 and 1 increases with increase in depth. The bar plot of comparison of accuracy and time consumption for TMDT and C4.5 when implemented in Python platform for the five datasets is shown in Fig. 6. It is observed that TMDT has much lower latency as compared to C4.5 while accuracy of TMDT is slightly lower as compared to C4.5 which becomes comparable with increase in dataset size. Thus TMDT gives comparable performance while running 28× faster than C4.5 in worst case comparison. The time consumption comparison for these 5 datasets for C, Python and hardware implementation (Verilog HDL) is reported in Table 1. The software implementations were found to have higher training latency as compared to the hardware implementation. The fastest implementation on C took 224 msec whereas FPGA implementation took only about 24 msec which is almost 10× faster for the largest dataset skin. Thus, the hardware implementation was observed to reduce the training latency further. The hardware utilisation comparison of the proposed training architecture with conventional and optimised k-means classification architecture is discussed in Table 2. This training accelerator hardware utilises only 6.488k FF and 0.228k BRAM for 38k data-instances as compared to the optimised k-means classification hardware proposed in [21] which utilised 24k FF and 0.24k BRAM for only 16.384k data-instances. But the LUT and DSP consumption is little higher in this design due to higher computational complexities as compared to classification. Thus dynamic power consumption for this design is 3W only.

Conclusion and Future Work

This paper proposes a 32-bit serial architecture running at 62 MHz to implement the training of TMDT algorithm on FPGA which provides re-configurability thus allowing dynamic training. As the proposed hardware is capable to test 5 binary data-sets each having different dimensions and size, it can be used to train wide variety of datasets. The TMDT algorithm implemented on this hardware runs at least 28× faster and has lower complexity than the widely used conventional C4.5 algorithm. This architecture have been shown to achieve at least 10× speed-up as compared to software implementations for same datasets by introducing

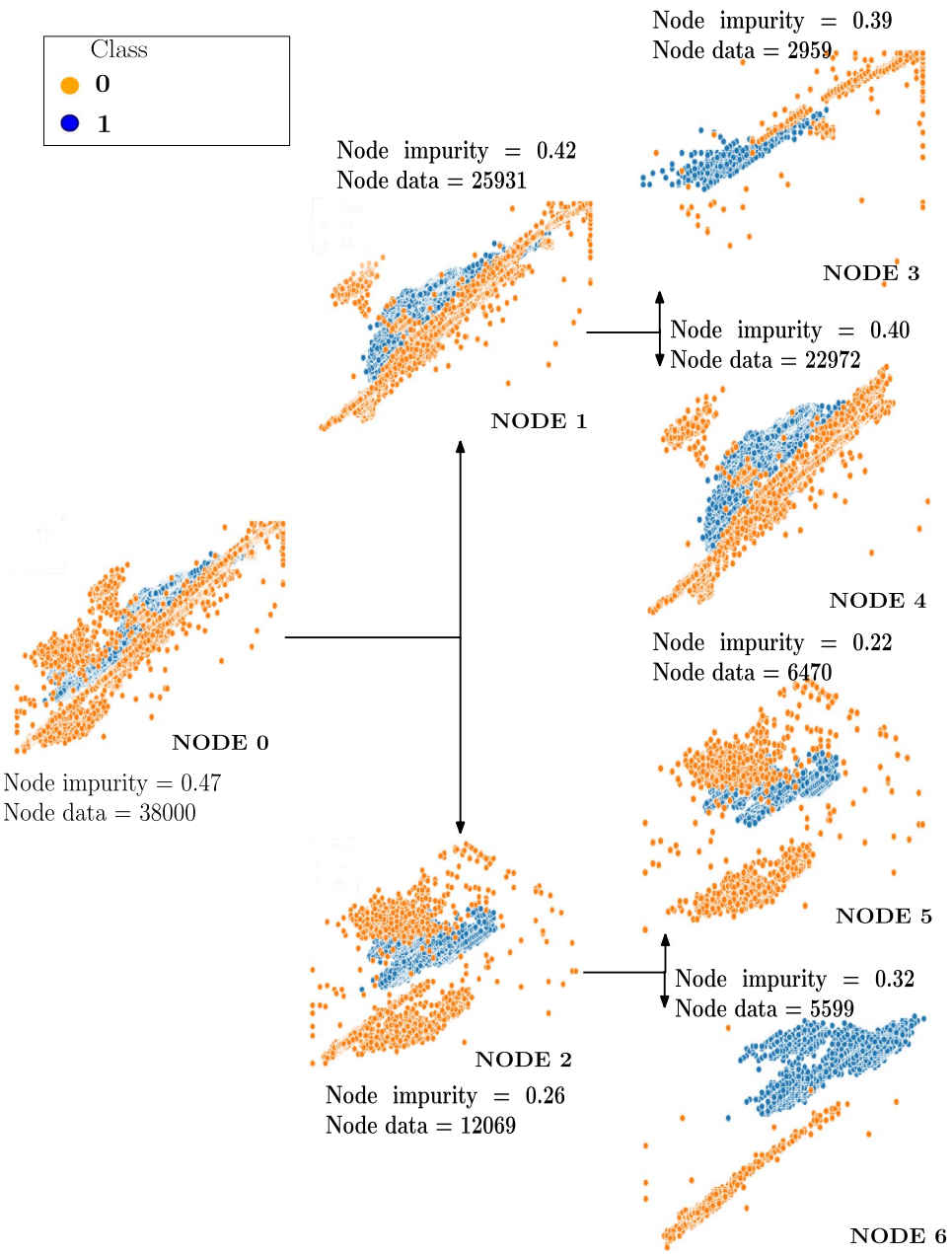

Fig. 5 Node data separation from parent node to children nodes in TMDT as visualised above for Skin dataset for the split nodes. The data overlap is gradually reduced from parent node to children nodes. The node impurity as indicated in the figure is observed to drop gradually with increase in depth

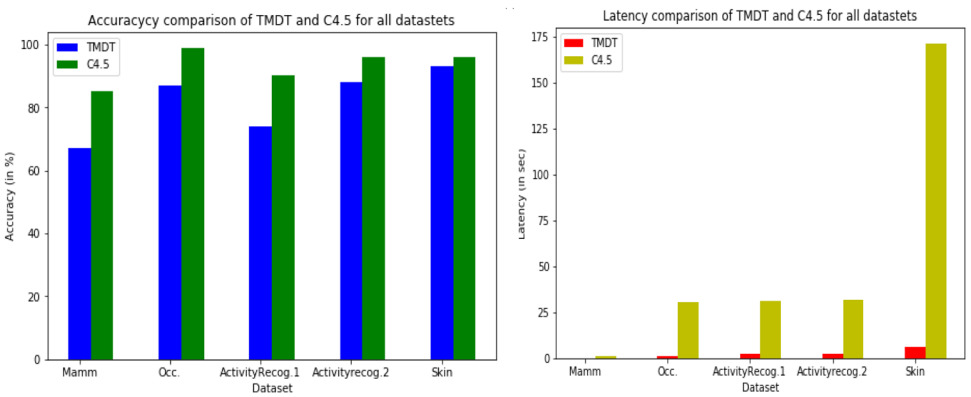

Fig. 6 Bar plots showing comparison of latency and accuracy for all datasets for C4.5 and TMDT algorithm

Table 1 Latency Comparison of Software and Hardware Implementation

TMDT Implementation Platform				Dataset	
	Mamm. (in ms)	Occupancy (in ms)	Activityrecog.1 (in ms)	Activityrecog.2 (in ms)	Skin (in ms)
Python	156	1111	2334	2530	6000
C	8.76	100.93	79.3	82.07	224.5
Verilog HDL	0.494	6.4	6.27	6.26	24
Data Dimension	793×6	10152×6	14389×6	14389×6	38000×4

Table 2 Resource utilisation comparison with existing design

	LUT (in K)	FF (in K)	BRAM (in K)	DSP (in K)	Data-points (in K)	FPGA Board	Algorithm implemented
k-means	108	54	0.100	1.062	16.384	Virtex 7	Conventional k-means classification
[21]	14	24	0.240	0.186	16.384	Virtex 7	Optimised k-means classification
Proposed design	894	6.488	0.228	0.2	38	Virtex Ultrascale+	TMDT training

some extent of parallelism. In the future works, the parallel and pipelined versions of the design will be implemented.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

Data availablility The data are available publicly at https://archive.ics.uci.edu/ml/index.php .

References

1. Ahmad MW, Mourshed M, Rezgui Y. Trees vs neurons: Comparison between random forest and ANN for high-resolution prediction of building energy consumption. Energy Build. 2017;147:77–89.
2. Behnke S, Karayiannis NB. Competitive neural trees for pattern classification. IEEE Trans Neural Netw. 1998;9(6):1352–69.
3. Buschjäger S, Morik K. Decision tree and random forest implementations for fast filtering of sensor data. IEEE Trans Circuits Syst I Regular Pap. 2017;65(1):209–22.
4. Canilho J, Véstias M, Neto H. Multi-core for k-means clustering on FPGA. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), IEEE; 2016, pp. 1–4.
5. Chrysos G, Dagritzikos P, Papaefstathiou I, Dollas A. HC-CART: A parallel system implementation of data mining classification and regression tree (cart) algorithm on a multi-fpga system. ACM Trans Archit Code Optim (TACO). 2013;9(4):1–25.
6. Davey N, Adams R, George SJ. The architecture and performance of a stochastic competitive evolutionary neural tree network. Appl Intell. 2000;12(1–2):75–93.
7. Dua D, Graff C. UCI machine learning repository; 2017. http://archive.ics.uci.edu/ml.
8. Estlick M, Leeser M, Theiler J, Szymanski J.J. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays; 2001, pp. 103–10 .
9. Foresti G.L, Dolso T. An adaptive high-order neural tree for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) (2004);**34**(2), 988–996.
10. Li T, Tang YY, Fang L. A structure-parameter-adaptive (spa) neural tree for the recognition of large character set. Pattern Recognit. 1995;28(3):315–29.
11. Lopez-Estrada S, Cumplido R. Decision tree based fpga-architecture for texture sea state classification. In: 2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006), IEEE; 2006; pp. 1–7.
12. Milone, D.H., Sáez, J.C., Simón, G., Rufiner, H.L.: Self-organizing neural tree networks. In: Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol. 20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No. 98CH36286), vol. 3, IEEE; 1998, pp. 1348–51.
13. Qian M. Application of CORDIC algorithm to neural networks VLSI design. In: The Proceedings of the Multiconference on" Computational Engineering in Systems Applications", vol. 1, IEEE;2006, pp. 504–8.
14. Quinlan JR. Induction of decision trees. Mach Learn. 1986;1(1):81–106.
15. Sahin S, Becerikli Y, Yazici S. Neural network implementation in hardware using FPGAs. In: International Conference on Neural Information Processing; 2006, pp. 1105–12.
16. Saqib F, Dutta A, Plusquellic J, Ortiz P, Pattichis MS. Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF). IEEE Trans Comput. 2013;64(1):280–5.
17. Shoaran M, Haghi BA, Taghavi M, Farivar M, Emami-Neyestanak A. Energy-efficient classification for resource-constrained biomedical applications. IEEE J Emerg Select Topics Circuits Syst. 2018;8(4):693–707.

18. Song HH, Lee SW. A self-organizing neural tree for large-set pattern classification. IEEE Trans Neural Netw. 1998;9(3):369–80.
19. Su MC, Lo HH, Hsu FH. A neural tree and its application to spam e-mail detection. Expert Syst Appl. 2010;37(12):7976–85.
20. Tong D, Qu YR, Prasanna VK. Accelerating decision tree based traffic classification on FPGA and multicore platforms. IEEE Trans Parallel Distrib Syst. 2017;28(11):3046–59.
21. Winterstein F, Bayliss S, Constantinides G.A. FPGA-based K-means clustering using tree-based data structures. In: 2013 23rd International Conference on Field programmable Logic and Applications, IEEE; 2013, pp. 1–6.
22. Yang Y, Boling C.S, Mason A.J. Power-area efficient VLSI implementation of decision tree based spike classification for neural recording implants. In: 2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings, IEEE; 2014; pp. 380–3.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.