From Arithmetic to Logic based AI: A Comparative Analysis of Neural Networks and Tsetlin Machine

Jie Lei†, Adrian Wheeldon†, *Member, IEEE*, Rishad Shafik†, *Senior Member, IEEE*,
Alex Yakovlev†, *Fellow, IEEE* and Ole-Christoffer Granmo‡
†Microsystems Research Group, School of Engineering, Newcastle University, NE1 7RU, UK.
‡Centre for AI Research (CAIR), University of Agder, Kristiansand, Norway.

***Abstract*— Neural networks constitute a well-established design method for current and future generations of artificial intelligence. They depends on regressed arithmetic between perceptrons organized in multiple layers to derive a set of weights that can be used for classification or prediction. Over the past few decades, significant progress has been made in low-complexity designs enabled by powerful hardware/software ecosystems.**

Built on the foundations of finite-state automata and game theory, Tsetlin Machine is increasingly gaining momentum as an emerging artificial intelligence design method. It is fundamentally based on propositional logic based formulation using booleanized input features. Recently developed Tsetlin Machine hardware architecture has demonstrated competitive performance and accuracy as well as opportunities for by-design energy efficiency and explainability.

In this paper, we investigate these two architectures closely and perform a comprehensive, comparative analysis considering their architectural subtleties implemented in low-level C language and ignoring any specialized implementations. We study the impact of hyperparameters on both arithmetic and logic basis of learning in terms of performance, accuracy and energy efficiency. We show that Tsetln Machine consistently outperforms artificial neural network in terms of learning convergence and energy efficiency by up to 15× at the cost of higher energy consumption per epoch.

***Index Terms*—Pervasive AI, Learning Automata, Tsetlin Machine, Artificial neural network, Machine Learning**

I. Introduction

Artificial intelligence (AI) is the fundamental enabler of fourth industrial revolution [1]. Empowering this technology pervasively can transform the way we perform information-led disease diagnosis [2], provide security by real-time image recognition [3] or predict financial frauds by identifying uncharacteristic patterns [4]. In realizing this technology, there are significant challenges surfacing around energy efficiency, design productivity and explainability – which are being extensively researched recently [5], [6].

Existing AI systems predominantly follow the principle of neural networks (NNs). Originally inspired by Rosenblatt's Neural Automaton in 1957 [7], modern NN implementations, such as artificial neural network (ANN), project the characteristics of human neurons into arithmetic operations to generalize the patterns of inputs. By diversifying the connections between a number of neurons the learning problem is reduced to a multi-path weighted sum of inputs. During training these weights are suitably adjusted by rigorous gradient descent exercises. Over the years, researchers have generated powerful hardware-software ecosystems as well as low-complexity NN design methods to address performance, accuracy and energy efficiency tradeoffs [8].

A new machine learning (ML) algorithm inspired by learning automata and game theory has recently been proposed by [9]. This algorithm, namely the Tsetlin machine (TM), is founded on propositional logic, moving away from the traditional arithmetic based ML underpinning. As such, it has demonstrated promising performance with improved hardware energy efficiency while also providing comparable accuracy [1]. Nevertheless, limited research has been carried out to investigate the fundamental difference between ANN and TM. Understanding this difference is crucial for developing powerful AI solutions targeting energy-frugality and explainability objectives.

This paper serves to fill this gap and presents a high-level comparative analysis of the ANN and TM as well as performance benchmarks using low-level (i.e. C language based) implementations and experiments. It is organized as follows: Sections II and III provide an overview of the architectures along with their hyperparameter search for design space exploration. Section IV demonstrates the architectural differences and presents a performance comparison using the following datasets: house voting record (HVR)[1], breast cancer (BC)[2], Sonar[3] and Modified National Institute of Standards and Technology (MNIST)[4]. These datasets are carefully chosen to represent classification diversity, inter-class correlations and complexity. Finally, Section V concludes the paper.

II. Artificial Neural Network

Figure 1 shows the block diagram of an ANN. It consists of multi-layer perceptrons that are able to infer or classify using the weighted sum of values through forward propagation. The $weights$ and $biases$ are updated by backpropagation through gradient descent exercises during training.

Table I provides the fundamental hyperparameters of the ANN. They can be divided into two types: 1) those that impact the capacity of the network ($layer$, $layer_{hid}$, $node_{hid}$) and 2) those that impact the efficacy of learning ($lrate$, $step$).

As shown in the Figure 1, the learning in ANN can be organized in the following steps. In the first step, hyperparameters are searched on ANN to set up the structure of the

[1]Congressional Voting Records: https://tinyurl.com/VotingDataset
[2]Breast Cancer (Diagnostic): https://tinyurl.com/BreastCancerDiagnDataset
[3]Sonar: https://tinyurl.com/ConnectionistBenchDataset
[4]MNIST: https://tinyurl.com/MNISTdatasets

Table I
HYPERPARAMETERS, THEIR SYMBOLS AND SYSTEM-WIDE IMPACT

Hyperparameters in ANN	**Symbol**	**Learnable**	**Tuneable**	**Impact**
Number of layers	$layer$		✓	Influences capacity of the network
Number of hidden layers	$layer_{hid}$		✓	Influences capacity of the network
Number of hidden nodes	$node_{hid}$		✓	Influences capacity of the network
Learning rate	$lrate$		✓	Influences step size of weights update
Types of activation functions			✓	Influences the product of neurons
steepness of activation function	$step$		✓	Influences output range of activation functions
Weights	w	✓		Decides how much the influence the inputs of neurons will have to the output
Biases	b	✓		Offsets the product of neurons
Hyperparameters in TM	**Symbol**	**Learnable**	**Tuneable**	**Impact**
Number of booleanized inputs	L		✓	Data prepossessing specific, may influences accuracy/energy/runtime
Number of clauses per class	U		✓	Significant influence on accuracy/energy/runtime
Number of automanton states	$2n$		✓	Influences reachability, less influence on accuracy/energy/runtime
Feedback threshold	T		✓	Influences probability of learning events in clauses
Learning sensitivity	s		✓	Influences probability of penalty/reward in automaton
Current automaton states	ST	✓		Influences formation of clause values, stores generalized patterns

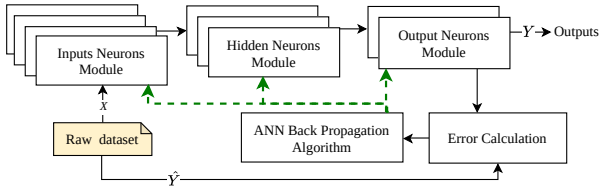

Figure 1. Block diagram of ANN

network based on the number of inputs and output of the dataset. The second step involves the choice of activation functions based on the learning steepness for the given dataset. For example, non-linear activation functions such as Leaky ReLU is recommended for hidden layers, while Sigmoid output activation function may work efficiently on dataset that has binary encoded outputs [10]. Significant variation in the steepness of the activation functions can easily cause gradient vanishing or exploding issue which causes learning inefficacy in ANNs [10]. In the third step, a grid search is performed on the number of hidden layers and nodes to match the capacity of the network with the complexity of hidden pattern. This step may reduce the complexity of the network and mitigate overfitting during learning, which often improves accuracy to specific training datapoints but does not generalize for unknown datapoints during testing. In the final step, the learning rate is fine-tuned to further increase the accuracy. It determines the steepness of weight update by controlling the rate of network convergence. Gradient descent instability is often common if this parameter is not carefully chosen.

Hyperparameter search on the ANN mainly involves time-consuming empirical experiments and grid search. Deeper ANNs are capable of handling more complex patterns but they are harder to train [11] due to the well-known gradient instability and large number of parameters [10].

III. TSETLIN MACHINE

TM is a promising ML algorithm based on propositional logic [9]. As shown in Figure 2, data encoding is the key differentiator in TM, as the input data are booleanized rather than binary encoded as in the ANNs. The key idea is to encode the data informational content as a set of Boolean features. These features constitute the *clause computation module* for both learning and inference. This module consists of a team of Tsetlin automata (TAs) with internal states (i.e. a cluster of finite state machines (FSMs)), which determine the *include* or *exclude* actions for the booleanized features and their complements (in total $2L$). A propositional logic expression generated from these actions within this module constitutes the Boolean output of a *Clause*.

During training, an ensemble of clause outputs are sent to *summation and threshold* module for classification. In each training datapoint (i.e. a set of booleanized training data), the TM classifications (Y) are compared with the known classes ($\hat{Y}$). The *feedback module* uses the differences between $\hat{Y}$ and Y to form the reinforcements, namely *penalty* or *reward*. The TAs will update their states as well as actions based on these reinforcements. The actions correspond to the patterns that the TM observed from the booleanized inputs and eventually forms a propositional logic within each *Clause* to determine its output. This training cycle can repeat until the training target is met. After training, testing can commence by disabling the *feedback module*. Classification or predictions of Y can be performed based on the Boolean literals and the pre-determined states and actions in each TA [9].

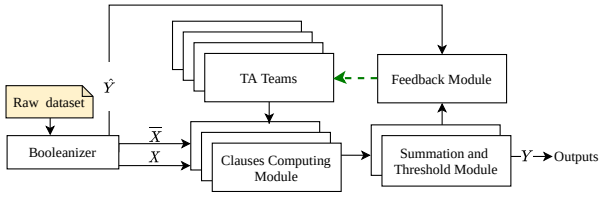

Figure 2. Block diagram of TM

Table I shows the TM hyperparameters. The most important hyperparameters in the TM are T and s. They control the frequency and probability of penalty or reward reinforcements in the TAs that states in Figure 5. Additionally, the learning capacity of the TM is determined by the number of clauses (U); the capacity grows exponentially with the increase of U at the expense of increased resource usage.

The number of booleanized features influences the efficiency of the TM dramatically since more input literals will result in more TAs participating in the clause value formation. Lastly, the number of states ($2n$) is also an important

parameter in the TM as it determines the complexity of the FSM. A designer must carefully consider the dataset size and choose hyperparameters accordingly to ensure a good tradeoff between learning efficacy and the overall energy consumption.

IV. Comparative Results and Analysis

A. Architectural Overview

Based on the architectural overviews in Section II and Section III, the key difference between the ANN and TM is data encoding. ANNs binarize the input data and process them arithmetically, while TMs booleanize data in order to generate the propositional logic expressions within each *Clause module*. Figure 3 and Figure 4 illustrate this difference in corresponding building blocks of the above ML algorithms.

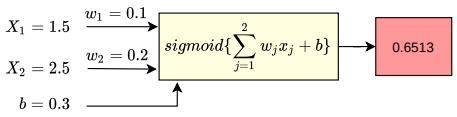

Figure 3. A schematic diagram of TM, showing different structural blocks

As shown in Figure 3 perceptron is the modular learning component in an ANN. The overall learning problem is broken into sequence of pereceptrons in multiple combinations of data paths, depending on the structural hyperparameters. Each path stochastically differs from the others due to connectivity between perceptrons and contributes in an overall regression exercise. The complexity of this regression exercise depends on the number of perceptrons and their connections. The more complex datasets typically have higher number of pereceptrons and connections. The regressed and weighted multi-path outcomes are accumulated at the output layer to compare against a classification threshold. During training, the weights are suitably updated in iterative learning steps until the desired classification accuracy is achieved.

In a TM, *Clause* is the modular learning component as shown in Figure 4. Each clause independently randomizes the states in a team of TAs depending on the reinforcement feedback and hyperparameters. The TA states determine the actions applied on the booleanized literals and eventually forms a propositional logic expression. This expression generates the *Clause* output, which in turn contributes to classification as well as the feedback process. As the dataset complexity or dimensionality increases more clauses are needed. However, as these are logically independent learning between clauses can be highly parallel.

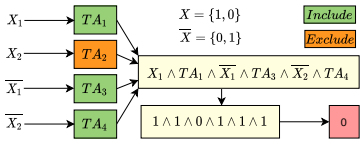

Figure 4. Example of clause computing module in TM

B. Comparative Analysis of Energy Efficiency and Accuracy

This section presents the comparative learning performance of the ANN and TM using the datasets: HVR, BC, Sonar and MNIST. These datasets aim to cover four categories of ML problems as follows: 1) classification with no overlap between classes (HVR), 2) minor overlaps between classes (BC), 3) moderate to strong class overlaps (Sonar) and 4) dataset with larger dimensionality (MNIST).

Table II compares experimental results of learning times (in epochs), runtime (in seconds), memory footprints and energy consumptions at target test accuracies. The ANN experiments are based on *FANN*[5], which is a simple and well-established multi-layer neural network implementation. The TM experiments are carried out using *TsetlinMachineC*[6] based on the original TM design. Low-level C implementations are chosen to demonstrate close-to-hardware and more reliable learning performance figures of the algorithms, essential for our memory and energy efficiency comparisons. The runtime, memory and estimated energy of the implementations were collected by software instrumentation tools: *likwid*[7] and *powerstat*[8].

The results on HVR and BC datasets indicate that TM has higher overall energy efficiency (by up to 15× in the case of HVR), faster runtime and lighter memory footprint compared to the ANN. The highly parallelized TM allows every TA receive independent and customized reinforcements to quickly generalize propositional logic patterns. The clear class separation in the datasets coupled with an independent logic based learning in clauses have contributed to these gains. However in the ANN, the learning requires the weights and biases to progressively adjust by backpropagation for preventing gradient instability [10]. The network repeats this process based on the learning error rate in variable-step arithmetic updates, while regressing on the previously known weights, thus takes longer time. Figure 5 and Figure 6 further reflect on this observation, as ANN learning process updates in gradient descent steps.

Although the TM consumes less energy and time on BC dataset, the energy per epoch is approximately 4× greater than that of ANN. The TM energy per epoch is affected by the number of input literals (X and $\overline{X}$) which increases from 64 on the HVR to 600 on the BC dataset. The more literals the TM has, the more TAs will participate in clause calculation in each datapoint. Typically, when the number of literals increases more clauses are required for higher learning accuracy. As such, when d datapoints are iterated in each epoch a total of $d \times U \times 2L$ reinforcement steps are required as opposed to maximum N weight updates, where N is the total number of perceptrons in the ANN. Our observations show that the clause formation contributes to ≈70% of the computation time.

Higher energy per epoch makes the TM sensitive to booleanization methods as demonstrated by the Sonar dataset. The number of input literals of the TM increases to 1652 on the Sonar dataset due to poorly designed booleanization (which generates extraneous boolean features). As a result, the TM spends more time and energy than the ANN although the TM still maintains higher rate of convergence. The ANN is approximately 14× faster and 17× more energy efficient than the TM in every epoch. Advanced significance-driven

[5]FANN: https://github.com/libfann/fann
[6]TsetlinMachineC: https://github.com/cair/TsetlinMachineC
[7]likwid: https://github.com/RRZE-HPC/likwid
[8]powerstat: https://launchpad.net/ubuntu/xenial/+package/powerstat

Table II
COMPARATIVE LEARNING PERFORMANCES OF ANN AND TM ON 4 DATASETS.

	ANN					TM				
Dataset	Epoch	Runtime	Memory	Test Accuracy	Energy	Epoch	Runtime	Memory	Test Accuracy	Energy
HVR	120	0.210 s	3.01 MB	95%	3.78 J	11	0.013 s	1.68 MB	95 %	0.25 J
BC	145	0.387 s	3.15 MB	95%	7.12 J	5	0.051 s	2.46 MB	95 %	1.07 J
Sonar	40	0.062 s	3.11 MB	80%	1.24 J	5	0.108 s	2.74 MB	80 %	2.59 J
MNIST	50	172.74 s	253.7 MB	92%	3304 J	15	50.89 s	218.1 MB	92 %	997 J

booleanization algorithms are essential to mitigate the high energy demand of the TM per epoch [1].

Compared to Sonar dataset, the booleanization of the MNIST dataset uses a simple thresholding function to normalize the input features to "1" or "0" if the grayscale of the pixel is larger or smaller than "75". This method captures the essential informational values of each pixel with minimal number of booleanized literals (i.e. 28×28 pixels generating two literals each and producing a total of 1568 literals). Such a minimalist booleanization enables higher energy efficiency against the ANN implementation [12] with the energy per epoch approximately the same between the two designs.

Across the four dataset implementations, the TM produces less memory footprints than the ANN. This is because TM uses logic variables as opposed to fixed/floating point variables in the case of ANN. This powerful feature can be exploited towards designing fast TM hardware architecture with lean memory footprints. ANN hardware architectures are widely known for memory walls for larger datasets as the number of weights can dramatically increase. As such, researchers have proposed a number of memory compaction methods which include in-memory computing and binarized neural networks.

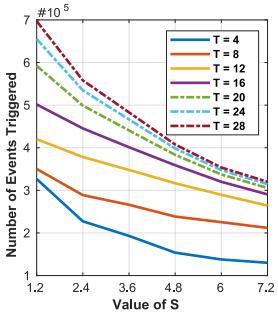

Figure 5. The affect of T and s on reinforcements on TM

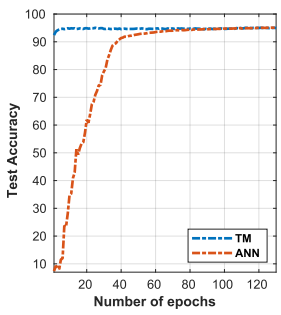

Figure 6. Test accuracy of ANN and TM on HVR dataset.

V. SUMMARY AND CONCLUSIONS

We presented a comparative analysis of ANN and TM considering the architectural differences, design space exploration with hyperparameter search and performance analysis on four different ML datasets. Our findings are summarized as below:
a) The TM has fewer hyperparameters which control the stochasticity during training and as such it features low-complexity design space exploration. ANNs have a number of sensitive hyperparameters such as learning rate and size of the network which results in gradient descent instability issues without careful balance between the hyperparameters [10].
b) The TM is built on modular clauses with propositional logic underpinning. As clauses are independent, the architecture lends itself to high parallelization which is conducive to energy-frugal hardware synthesis [1]. Although the low-level C implementation did not exploit this in software, but comparisons demonstrate higher performance for the TM due to low-complexity logic routines. The ANN heavily relies on complex arithmetic operations in perceptrons as the modular components. As such it requires more resources in computation as well as memory storage (for frequent weight updates).
c) The learning performance comparisons demonstrate high convergence of TM on all datasets; however, the energy per epoch decreases with the increase of number of input literals. Therefore, care must be exercised in designing the booleanized literals to minimize the total number of literals for a given ML problem. Due to arithmetic-heavy learning routines with variable-step gradient descent ANN convergence is typically slow and depends on the tradeoff between learning rate, the number of perceptrons and their connections.

Our future works include comparing their performance differences and studying the impact of stochasticity in learning using hardware implementations such as ASIC by fully exploiting their natural parallelism.

Acknowledgement: The authors gratefully acknowledge the funding from EPSRC IAA project "Whisperable" and EPSRC grant STRATA (EP/N023641/1).

REFERENCES

[1] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, "Learning automata based AI hardware design for IoT," *Philosophical Trans. A of the Royal Society*, 2020.
[2] J. Latif, C. Xiao, A. Imran, and S. Tu, "Medical imaging using machine learning and deep learning algorithms: A review," in *iCoMET*, 2019.
[3] L. C. Jain, U. Halici, I. Hayashi, S. B. Lee, and S. Tsutsui, Eds., *Intelligent Biometric Techniques in Fingerprint and Face Recognition*. CRC Press, Inc., 1999.
[4] M. Wu, Y. Huang, and J. Duan, "Investigations on classification methods for loan application based on machine learning," in *ICMLC*, 2019.
[5] R. Shafik, A. Yakovlev, and S. Das, "Real-power computing," *IEEE Transactions on Computers*, 2018.
[6] R. Shafik, A. Wheeldon, and A. Yakovlev, "Explainability and dependability analysis of learning automata based AI hardware," in *IEEE IOLTS*, 2020.
[7] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
[8] L. Benini, "Plenty of room at the bottom: Micropower deep learning for cognitive cyberphysical systems," *RTNS Keynote*, 2017.
[9] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv*, Apr. 2018.
[10] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *ICSCC*, 2019.
[11] X. Liu, J. Zhou, and H. Qian, "Comparison and evaluation of activation functions in term of gradient instability in deep neural networks," in *CCDC*, 2019.
[12] R. K. Mohapatra, B. Majhi, and S. K. Jena, "Classification performance analysis of mnist dataset utilizing a multi-resolution technique," in *ICCCS*, 2015.