Multi-Layer Tsetlin Machine: Architecture and Performance Evaluation

Olga Tarasyuk [1,2]
[1] *School of Engineering, Newcastle University*
Newcastle upon Tyne, UK
[2] *Odesa Technological University STEP*
Odesa, Ukraine
https://orcid.org/0000-0001-5991-8631

Anatoliy Gorbenko [3,4]
[3] *School of Built Environment, Engineering and Computing*
Leeds Beckett University, Leeds, UK
[4] *National Aerospace University "KhAI"*
Kharkiv, Ukraine
https://orcid.org/0000-0001-6757-1797

Alex Yakovlev
School of Engineering, Newcastle University
Newcastle upon Tyne, UK
https://orcid.org/0000-0003-0826-9330

Rishad Shafik
School of Engineering, Newcastle University
Newcastle upon Tyne, UK
https://orcid.org/0000-0001-5444-537X

***Abstract*—Tsetlin Machine (TM) is a recent automaton-based algorithm for reinforcement learning. It has demonstrated competitive accuracy on many popular benchmarks while providing a natural interpretability. Due to its logical underpinning, it is amenable to hardware implementation with faster performance and higher energy efficiency than conventional Artificial Neural Networks. This paper introduces a multi-layer architecture of Tsetlin Machines with the aim to further boost TM performance via adoption of a hierarchical feature learning approach. This is seen as a way of creating hierarchical logic expressions from original Boolean literals, surpassing single-layer TMs in their ability to capture more complex patterns and high-level features. In this work we demonstrate that multi-layer TM considerably overperforms the single-layer TM architecture on several benchmarks while maintaining the ability to interpret its logic inference. However, it has also been shown that uncontrolled growth in the number of layers leads to overfitting.**

Keywords—Machine learning, logic-based artificial intelligence, learning automaton, Tsetlin Machine, multi-layer architecture, performance, interpretability.

I. Introduction

Machine learning is an essential tool for analyzing complex data and making predictions in various fields, including finance, healthcare, and technology. Artificial neural networks have been the dominant machine learning technique in the recent years due to their high accuracy in predictive tasks. While Artificial Neural Networks (ANNs), especially deep neural networks (DNNs) have been successful in many application domains, they have certain limitations, such as high complexity and lack of interpretability. These limitations have led researchers to explore alternative machine learning methods, including the Tsetlin machine [1].

Tsetlin Machine (TM) is an emerging machine learning algorithm proposed by Granmo in 2018 [2]. The algorithm is underpinning on Tsetlin Automaton (TA) that originates from the research on collective behavior of learning automata by Tsetlin [3] and Varshavsky [4] and further developed in [5].

Recent studies have shown that TM provides a promising alternative to DNNs with several advantages: TM is an interpretable and low-complexity algorithm. It has a unique logic-based learning mechanism supporting parallelism and efficient hardware implementation that makes TM attractive for embedded applications and hardware acceleration [6, 7]. TM has been successfully used in various applications domains aimed at pattern recognition such as image classification [6], text categorization [8, 9], speech recognition and audio keyword spotting [10], intrusion detection [11], etc. Unlike ANNs, TMs do not require gradient descent, which makes them a quicker-converging learning algorithm. Moreover, TMs have considerably fewer number of hyperparameters to tune and are highly interpretable because their model prediction is carried out via single-layer propositional logic clauses. The last factor is crucial for safety and mission-critical applications.

TM has been actively developed over the last few years and has demonstrated competitive accuracy on different benchmarks [12]. There is also growing interest in building larger, more complex TM models and hierarchical capability such as those in DNNs.

The standard TMs produce variably sized conjunctions from a single layer followed by summative voting. While these have produced competitive accuracies, we are keen to explore the potential of multi-layered TMs that can provide hierarchical conjunctions followed by summative voting. This type of layering can also help in understanding the impact of conjunction clausing on the accuracy and performance of a new type of Tsetlin machines and their ability to learn complex patterns.

In this paper, we introduce a novel multi-layer architecture of Tsetlin Machines with the aim to reinforce TM performance via adoption of a hierarchical feature learning. Through this work we provide answers to the following research questions:

Q1: How to organize effective and resource-efficient multi-layer TM training?
Q2: How does the number of layers affect TM pattern recognition capability, i.e. training and validation accuracy?
Q3: How does the multi-layer TM learn patterns and how to interpret logical clauses constructed by each layer?

II. Basics of Tsetlin Machine

The theory on which the Tsetlin Machine is based originates from the research on collective behaviour of learning automata by Tsetlin [3]. Tsetlin studied how rats found their way in mazes (see Fig. 1) to determine how biological systems learn. He demonstrated that finite automata provided a sufficiently accurate and yet simple mathematical model for this behaviour. Such learning automata with linear tactics were later allocated to a special class called Tsetlin automata (TA) [13].

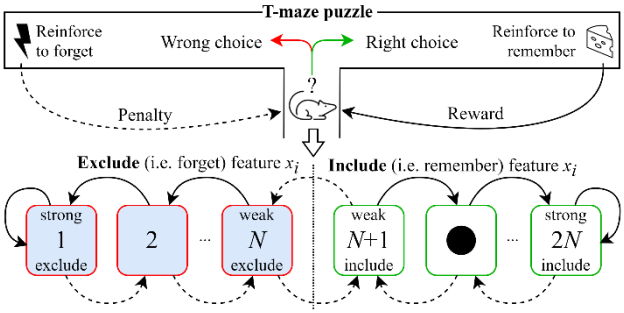

Fig. 1. Tsetlin Automaton – a mathematical representation of a rat brain: states and transitions.

The TA behaves like a finite state machine where $2N$ states are divided equally into two actions: to 'remember' or to 'forget' a given decision. Reward and penalty signals transition the TA between these states which allows it to remember the right choice deeper or forget the wrong one as shown in Fig. 1.

Granmo incorporated propositional logic and a game-theoretic bandit driven approach, namely the Tsetlin Machine, to orchestrate the collective behaviour of TAs and formulate logic statements [2]. Hundreds of such 'rat brains' (i.e., TAs), each making one decision, combined through NOT and AND operations to form logic propositions and used to solve classification tasks are referred to as the Tsetlin Machine.

Fig. 2 presents a high-level architecture and the core building blocks of the Tsetlin Machine.

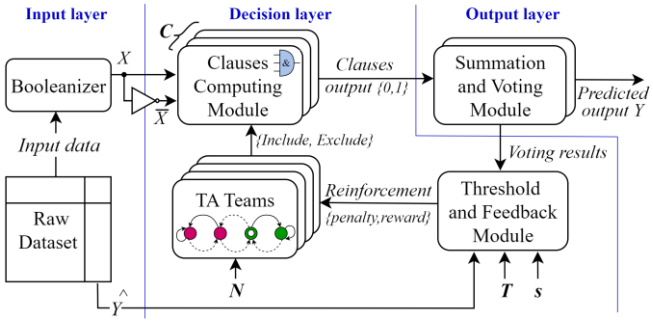

Fig. 2. High-level TM architecture.

Input Layer. TM accepts input variables as Boolean features (literals X and their negations $\bar{X}$). Booleanization means reducing the values of input variables to Boolean values (e.g. through simple ranking or thresholding methods or using more advanced significance-driven Booleanization algorithms [14]) so that each of them is either True or False.

Decision Layer. The learning process for the TM centers on building Boolean logic expressions – conjunctive clauses. The clause relates the Boolean literal to its respective TA to learn the decision regarding inclusion (i.e. remembering) or exclusion (i.e. forgetting) of that feature into/from the conjunctive clause corresponding to a particular class.

Each TA outputs '1' if its state is 'include' and '0' otherwise.

For a classification problem with M classes, there are C clauses, which are employed per class. Half of them is assigned to be positive, i.e. to learn distinctive class features from the class samples; the other half are negative clauses trained to recognize data samples that do not belong to the class. Each clause contains its own set of TAs (referred to as TA teams). The number of clauses used per class C and the number of TA states $2N$ are the two hyper-parameters defining the TM architecture and the model size.

Output Layer. The outputs from each logic clause (which could be 0 or 1) are referred to as votes. They are summed together and clipped by a threshold for each class. Negative clauses are summed with a negative sign. The class with the highest sum of votes v is the predicted classification.

The voting threshold T and the learning sensitivity s are other two TM hyper-parameters affecting the learning. During TM training, feedback is issued based on the expected class and the clause outputs generated. The feedback module compares the expected class for the evaluated data instance with the clause outputs and generates reward and penalty signals to each TA according to rules described in [2].

Training and Feedback. During training TM descends feedback from the expected class level to the clause level where the T hyper-parameter is used, and then to the TA level where the s parameter is used.

At the clause level the T hyper-parameter is bounded between 0 and the number of clauses in the class. When the clauses voting sum v is less than the user-set target T, they are reinforced with the probability $(T\text{-}v)/2T$. This hyper-parameter is used to control the probability with which a particular clause will be given feedback at the TA level. The s hyper-parameter may take any floating-point value above 1. This parameter is used at the TA level where its value is used to control the probability with which a TA will transition between states as $(s\text{-}1)/s$.

III. Multi-layer TM Architecture and the Hierarchical Training Aproach

Fig. 3 shows the proposed multi-layer TM (MTM) architecture, implementing the concept of hierarchical feature learning. This was inspired by success of deep neural networks in achieving state-of-the-art results in a wide range of applications, including image and speech recognition, natural language processing, etc.

However, unlike the multi-path multi-level regression used in DNNs, we put forward the idea of organizing conjunctive clauses into hierarchical patterns rather than single-layer ones, followed by summative voting.

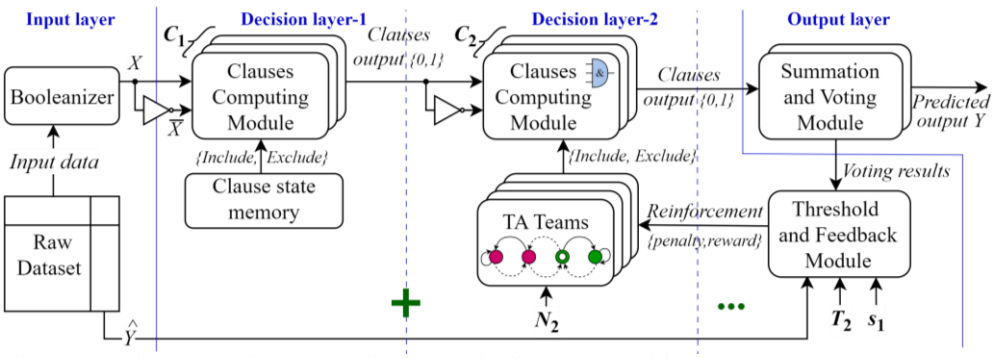

Fig. 3. Multi-layer TM architecture: two-layer example (clause outputs of the first-layer TM are used as input for the second-layer TM; after the first-layer TM is trained, its clause states are frozen by replacing TA team with the clause state memory; in turn, TA team is considered as a shared resource and is used to train the second-layer TM).

TABLE I. DATASETS

Dataset	Number of classes	Image size	Training samples	Validation samples
Digits-MNIST	10	28×28	60000	10000
Letters-MNIST	26	28×28	62400	10400
Fashion-MNIST	10	28×28	60000	10000
Kuzushiji-MNIST	10	28×28	60000	10000

The proposed architecture is composed of multiple layers of propositional clause logic. The first-layer TM uses Booleanized features (literals) of the original dataset as an input. Clause outputs of the first-layer TM are then used as a Boolean input to the second layer TM and so on.

In contrast to ANN, TM does not use back-propagation mechanism. During training, feedback is issued based on the expected class and the clause outputs generated. As a result, MTM can implement *a hierarchical training approach* where TMs of different layers can be trained sequentially starting from the first layer. It was previously shown that TM is characterised by the high rate of convergence [12]. Classification accuracy increases rapidly and quickly reaches near-the-maximum (for the given number of clauses and other hyperparameters) level. Further training leads to only a very incremental accuracy gain. Thus, when the first-layer TM reaches this stage, its training can be stopped and the state of propositional clauses frozen. After that, the second-layer TM can start its training by using the outputs of the first-layer TM clauses as input. The same process is repeated hierarchically for all subsequent layers.

When TMs of different layers are trained sequentially, the array of Tsetlin Automata can be treated as a shared resource and reused at each layer. This can considerably save hardware resources and increase efficiency of MTMs as compared to DNNs. Our further experiments, presented in Section IV, also show that the proposed hierarchical training approach allows us to achieve higher MTM performance compared to when TMs of all layers are trained in parallel.

IV. EXPERIMENTAL RESULTS

In this section, we will validate the effectiveness of the MTM approach proposed in Section III depending on the number of layers and the number of clauses in the subsequent layers.

A. Datasets

The paper explores pattern recognition performance of multi-layer TM architecture using several MNIST datasets (see in Table I). These datasets are one the most popular and widely recognized AI benchmarks for image recognition machine learning algorithms, the visual nature of which helps us to provide insights into MTM clause interpretability through as shown in Section V.

B. Performance of the Multi-layer TM Depending on the Number of Layers

The first set of experiments aims to confirm our hypothesis that adding extra layers to the TM architecture improves its classification accuracy.

TM accuracy largely depends on the amount of available hardware resources. These resources are mostly determined by the number of used Tsetlin Automata. The overall number of TAs depends on the number of logic clauses C that the TM allocates to each class (half of these clauses are marked as positive and are used to vote for the class; the other half are negative clauses intendent to vote against the class) multiplied by the number of classes and the number of Boolean literals (data features) which depend on the dataset. Positive clauses of each class form independent teams that learn persistent class patterns from data samples belonging to their class, without sharing knowledge between teams. Feeding the output of first-layer clauses into the Boolean inputs of the second layer (and so on) provides access for second-layer clauses to the knowledge accumulated by *all* first-level clauses of *all* classes. This hierarchical knowledge exchange increases information awareness from layer to layer, which should facilitate accurate decision making despite the limited number of clauses at each layer.

Previous experiments [15] suggest that using one hundred of clauses per class allows TM to achieve competitive accuracy on various MNIST datasets still leaving room for further improvement. The values of other two TM hyperparameters (the voting threshold T and the learning sensitivity s) were set as recommended in [15]:

1) the optimal voting threshold T approximates to the square root of the half of the number of clauses C; according to [16, 17] this ensures maximum voting power for each clause;

2) the optimal learning sensitivity s has logarithmic dependence on the number of clauses C and needs to be increased to achieve better selectivity in datasets with high inter-class similarity [15].

Tables II and III report training and validation accuracy of the multi-layer TM on different MNIST datasets depending on the number of layers. The tables also indicate the values of TM hyperparameters, which were set identical across all layers.

TMs of different layers were trained sequentially following the hierarchical training approach. Each layer, starting with the first one, was trained for 100 epochs until the TM accuracy began to saturate. The clause states were then frozen and used to generate training data for the next TM layer. We also continued individual training of TMs to be able to compare their accuracy after an equal number of training epoch. The learning dynamic of different layers of a multi-layer TM is shown in Fig. 4.

TABLE II. MULTI-LAYER TM TRAINING ACCURACY DEPENDING ON THE NUMBER OF LAYERS

Training accuracy / Dataset	Layer 1	2	3	4	5	MTM hyperparameters (C, T, s)
Digits-MNIST	97.72	99.16	99.67	99.85	99.93	(100, 8, 7.0)
Letters-MNIST	89.47	95.37	96.60	96.86	97.25	(100, 8, 7.0)
Fashion-MNIST	88.92	90.65	91.40	92.16	92.41	(100, 8, 10.2)
Kuzushiji-MNIST	94.37	97.06	98.11	98.69	98.87	(100, 8, 8.6)

TABLE III. MULTI-LAYER TM VALIDATION ACCURACY DEPENDING ON THE NUMBER OF LAYERS

Validation accuracy / Dataset	Layer 1	2	3	4	5	MTM hyperparameters (C, T, s)
Digits-MNIST	96.58	97.42	97.22	97.10	96.90	(100, 8, 7.0)
Letters-MNIST	85.30	88.96	88.85	88.06	87.58	(100, 8, 7.0)
Fashion-MNIST	86.56	86.84	86.82	86.76	86.63	(100, 8, 10.2)
Kuzushiji-MNIST	82.41	84.71	84.33	83.91	83.13	(100, 8, 8.6)

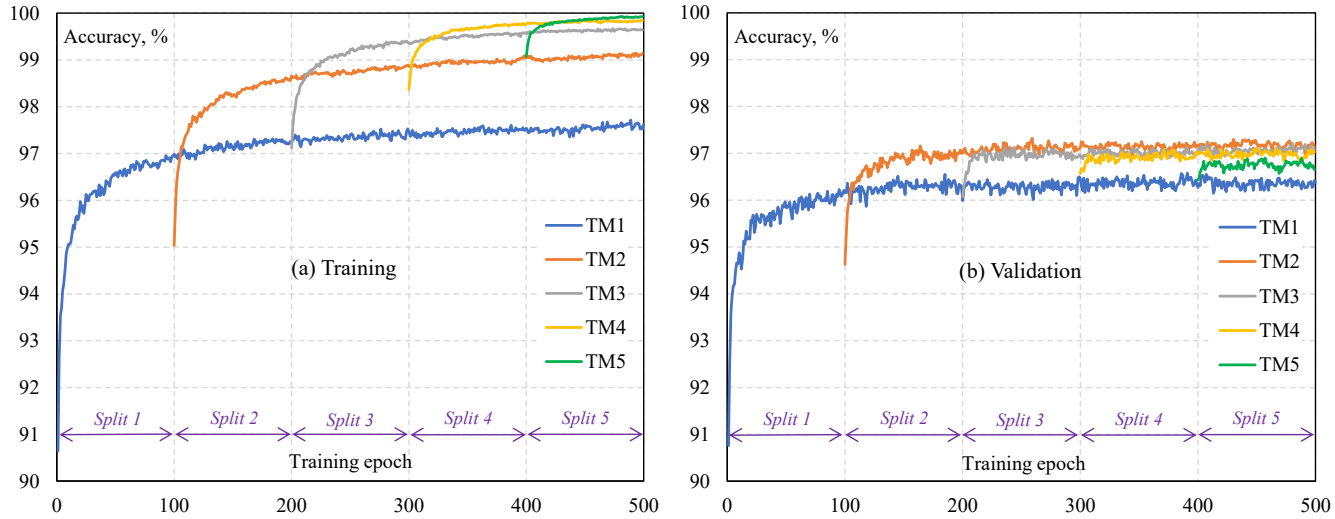

Fig. 4. Multi-layer TM accuracy on Digits MNIST dataset depending on the number of layers for the split training (the split size = 100 epochs): (a) training accuracy; (b) validation accuracy.

Experimental results suggest that the multi-layer approach can considerably improve TM performance. This is especially noticeable for the training accuracy, which reaches almost 100% on the Digits MNIST dataset with only 100 clauses in each of the five layers after only 500 training epochs in total. Although one can notice that the accuracy gain decreases from epoch to epoch.

The validation results (see Table III and Fig. 4,b) show that multi-layer TM is prone to overfitting issue, just like DNN. While the training accuracy continues to increase steadily from layer to layer, the validation accuracy from the third layer onwards begins to return to single-layer TM accuracy after an initial rapid increase. This process, observed in all used datasets, suggests that the two layers may be optimal for MTM generalization ability.

The hierarchical training approach allows to consider an array of Tsetlin Automata as a shared resource which can be reused at each layer as discussed in Section III. The number of epochs in each training cycle (i.e. training split) can be considered as an additional hyperparameter. Transition to next layer training should be performed after accuracy achieved at the current layer begins to saturate. Reducing the split size down to one epoch (see Fig. 5) reduces the overall efficiency of the multi-layer TM architecture.

C. *Performance of the Two-layer TM Depending on the Number of Clauses in the Second Layer*

As mentioned earlier, the number of logic clauses C that TM allocates to each class determines the amount of hardware resources. Minimizing used resources is important for applying machine learning at the edge. This section examines the two-layer TM architecture and analyses the extent to which its performance depends on the number of clauses in the second layer.

Used hyperparameters and experimental results obtained for different datasets are summarised in Tables IV-VI. Fig. 6 also shows the training dynamics of a two-layer TM on the Digits-MNIST dataset. It is shown that TM of the second layer does not necessarily need to have the same number of clauses as the first-layer TM to improve classification performance. Depending on the dataset, the accuracy gain can be achieved for as little as 20% of the clauses used in the second layer compared to the first layer. More complex datasets such as Fashion- and Kuzushiji-MNIST, which are characterized by high intra-class heterogeneity and/or inter-class similarity, require using up to 50-60% of the clauses at the second layer to boost TM performance. In general, the more clauses used in the second layer, the higher the training and validation accuracy.

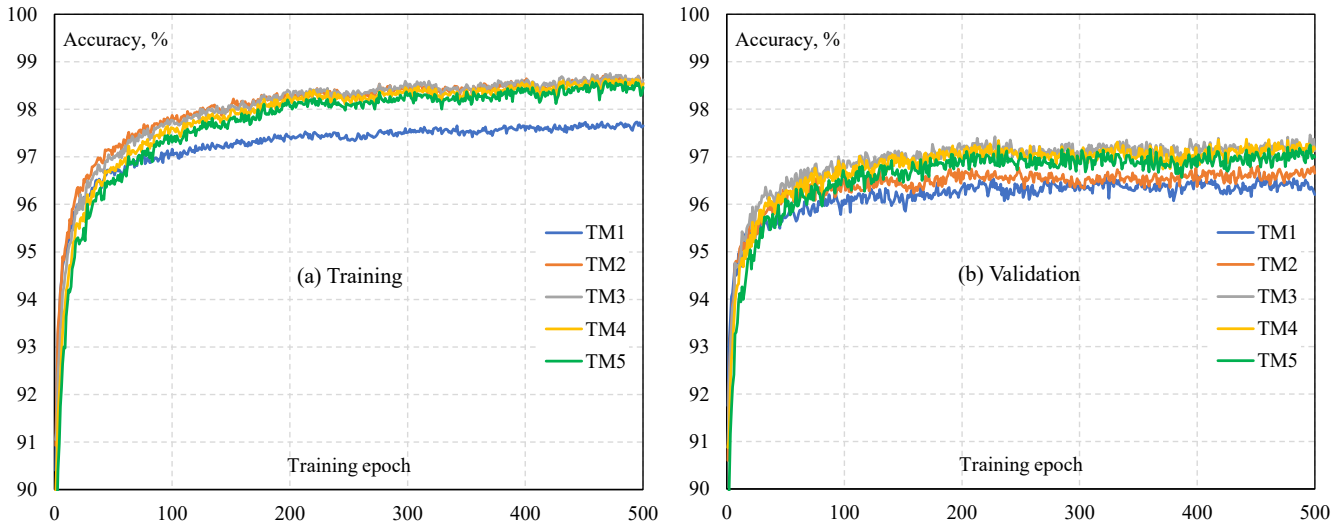

Fig. 5. Performance of the multi-layer TM on Digits MNIST dataset depending on the number of layers for the hierarchical micro-split training (the split size = 1 epoch): (a) training accuracy; (b) validation accuracy.

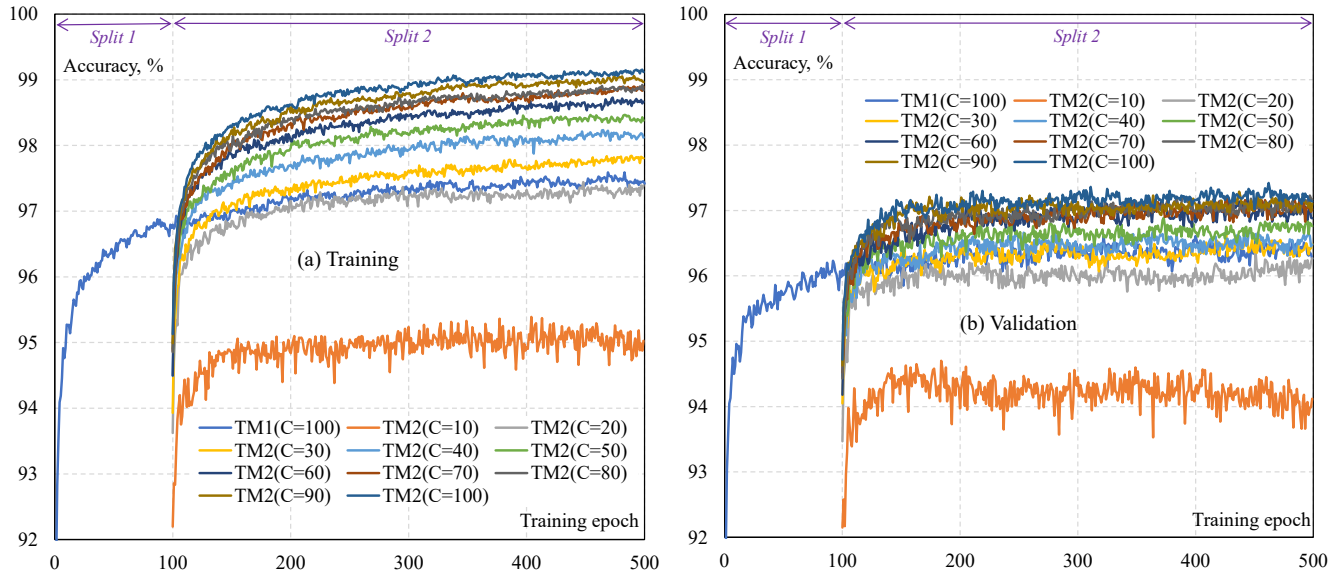

Fig. 6. Performance of the two-layer TM depending on the number of clauses in the second layer: (a) training accuracy; (b) validation accuracy.

V. Interpretability of the Multi-layer TM

In this section, we discuss how MTM creates hierarchical conjunctions of propositional logic and use visualization techniques to interpret the logical rules represented by TM clauses at different layers.

A. Impact of the First-Layer TM Clauses

Each positive TM clause of the first-layer TM can be considered as a class pattern (or class template) constructed by generalizing the class samples randomly selected from the dataset. Visualisation results suggest that the *s* hyperparameter plays a key role in building of such class patterns.

Fig. 7 depicts positive clauses of Class "0" for different *s*-values after training a TM with 20 clauses per class on the Digits-MNIST dataset. When the learning sensitivity *s* is large, TM clauses are more specific. When *s* gets smaller, each clause begins to generalize more class samples, resulting in "broadening" of clause templates. It is also worth to notice that when *s* is large, TM accuracy remains considerably low as true predictions can only be made for a small subset of data samples covered by *narrowly specialised* clauses. Smaller *s* boosts *generalization ability* of clauses. However, when clause templates become too general, they begin to pass through data samples belonging to other classes and, hence, produce false positive predictions.

Examples of Class "0" negative clauses used by TM to recognise data samples belonging to other classes and hence vote against Class "0" are shown in Fig. 8. Green colour (■) in Figs. 7 and 8 means inclusion of the Boolean feature x_i (i.e. inclusion of a black pixel from the original black-and-white image); read colour (■) means inclusion of the negated Boolean feature $\bar{x}_i$ (i.e. inclusion of a white pixel); white colour (□) means exclusion of both x_i and $\bar{x}_i$.

TABLE IV. Hyperparameters Used for the 1st and 2nd Layer TM

Hyperparameters (*T*, *s*) / Dataset	1st Layer	2nd Layer									
	C^1=100	C^2=10	C^2=20	C^2=30	C^2=40	C^2=50	C^2=60	C^2=70	C^2=80	C^2=90	C^2=100
Digits-MNIST	(8, 7.0)	(2, 1.5)	(3, 2.4)	(4, 3.6)	(4, 4.4)	(5, 5.0)	(5, 5.5)	(6, 6.0)	(6, 6.4)	(7, 6.7)	(8, 7.0)
Letters-MNIST	(8, 7.0)	(2, 1.5)	(3, 2.4)	(4, 3.6)	(4, 4.4)	(5, 5.0)	(5, 5.5)	(6, 6.0)	(6, 6.4)	(7, 6.7)	(8, 7.0)
Fashion-MNIST	(8, 10.2)	(2, 2.0)	(3, 3.1)	(4, 4.9)	(4, 6.1)	(5, 7.1)	(5, 7.9)	(6, 8.6)	(6, 9.2)	(7, 9.7)	(8, 10.2)
Kuzushiji -MNIST	(8, 8.6)	(2, 1.8)	(3, 2.8)	(4, 4.3)	(4, 5.3)	(5, 6.1)	(5, 6.7)	(6, 7.3)	(6, 7.8)	(7, 8.2)	(8, 8.6)

TABLE V. Two-layer TM Training Accuracy Depending on the Number of Clauses in the 2nd Layer

Training accuracy / Dataset	1st Layer	2nd Layer									
	C^1=100	C^2=10	C^2=20	C^2=30	C^2=40	C^2=50	C^2=60	C^2=70	C^2=80	C^2=90	C^2=100
Digits-MNIST	97.72	95.39	97.44	97.84	98.23	98.47	98.73	98.90	98.92	99.06	99.16
Letters-MNIST	89.47	86.43	90.44	91.90	92.35	93.60	93.84	94.70	94.91	95.14	95.37
Fashion-MNIST	88.92	87.18	87.93	88.62	89.02	89.42	89.75	89.98	90.30	90.32	90.65
K-MNIST	94.37	88.34	92.26	93.45	94.09	95.28	95.43	95.90	96.32	96.32	97.06

TABLE VI. Two-layer TM Validation Accuracy Depending on the Number of Clauses in the 2nd Layer

Validation accuracy / Dataset	1st Layer	2nd Layer									
	C^1=100	C^2=10	C^2=20	C^2=30	C^2=40	C^2=50	C^2=60	C^2=70	C^2=80	C^2=90	C^2=100
Digits-MNIST	96.58	94.70	96.36	96.63	96.70	96.89	97.19	97.21	97.24	97.29	97.42
Letters-MNIST	85.30	82.24	85.69	86.28	87.29	87.67	88.19	88.55	88.77	88.86	88.96
Fashion-MNIST	86.56	85.13	85.81	85.93	86.12	86.50	86.53	86.60	86.71	86.78	86.81
Kuzushiji -MNIST	82.41	72.73	77.94	79.82	80.80	82.53	82.61	83.01	83.55	83.55	84.71

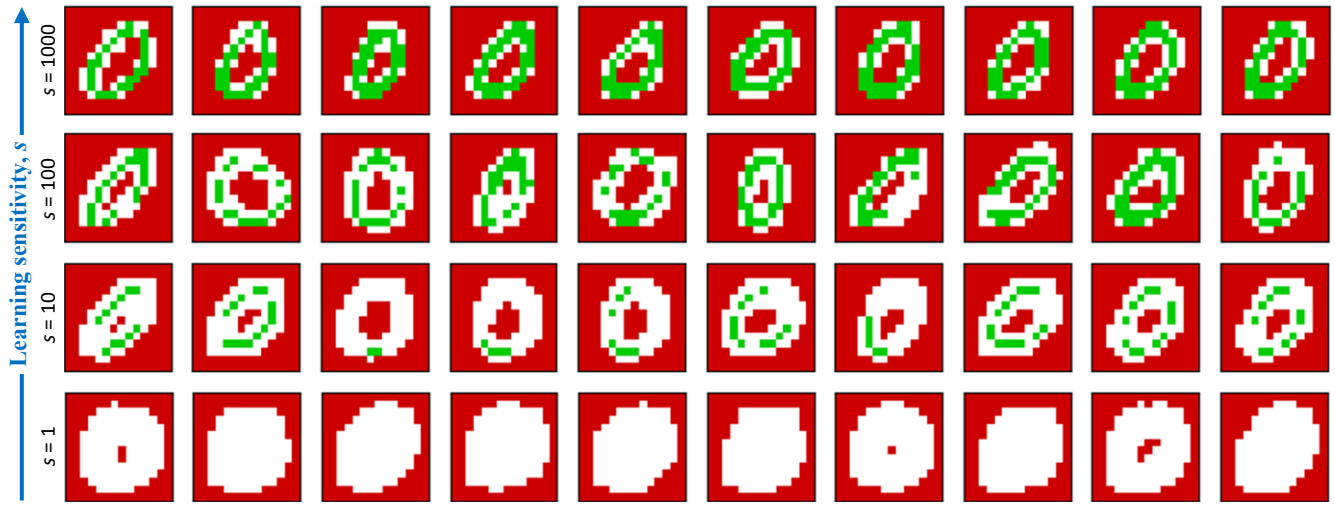

Fig. 7. Visualisation of TM (C=20, T=3, s=1..1000) *first-layer positive clauses* of Class "0" after training on the Digit MNIST dataset; each positive clause represents a certain template of Class '0'; the smaller the value of s, the wider the clause pattern.

Inclusion or exclusion of the certain Boolean literal depends on the state of the Tsetlin Automaton linked to this literal: if TA is in include state, it outputs *True*; otherwise – *False*. In general, each TM clause at the first layer can be represented as a conjunction of $(x_i \vee \neg TA_i^{L1})$ and $(\neg x_i \vee \neg TA_{2k+i}^{L1})$ disjunctions:

$$Clause_{Class=0..9,j=0..C-1}^{Layer1} = \\ = \bigwedge_{i=0}^{k-1} (x_i \vee \neg TA_i^{L1}) \wedge (\neg x_i \vee \neg TA_{2k+i}^{L1}),$$

where x_i – is a Boolean feature (e.g. a single pixel of a black-and-white image from the booleanized MNIST dataset), k – is the total number of Boolean features (e.g. pixels).

B. *Impact of the Second-Layer TM Clauses*

The stochastic feedback mechanism ensures that during training TM clauses randomly cover data samples with equal probability, provided that T hyperparameter is set optimally (too large T-value reduces clause diversity [15]). However, the first-level TM clauses can be considered as autonomous inference units that interact only during the voting process. They do not directly exchange or share knowledge (e.g. about covered class samples) with clauses of the same class or with clauses of other classes.

Second-layer TM does not see the original training data samples. Instead, it observes clause outputs coming from the first-layer TM. Thus, each second-layer clause benefits from seen the reaction of ALL (positive and negative) first-layer TM clauses of ALL classes (not only clauses of the same class) to each sample of input data. This extended awareness allows for more accurate predictions.

Examples of the second-layer positive and negative clauses are shown in Figs. 9 and 10. Their elements are the first-layer positive and negative clauses of all classes, the icons of which can be outlined with a green or blue border. The green frame around the icon indicates that the corresponding first-layer clause is included in the second-level proposition. The blue frame means that the second-layer clause includes the negation of the first-layer clause. The white box with a dashed border indicates the exclusion of both the corresponding first-layer clause and its negation.

Since TM only uses conjunctive clauses of propositional logic, positive clauses of the second layer cannot directly include positive clauses of the same class from the first layer (as it was shown earlier, each positive clause of the first layer TM generalises only some subset of class samples, but not all of them).

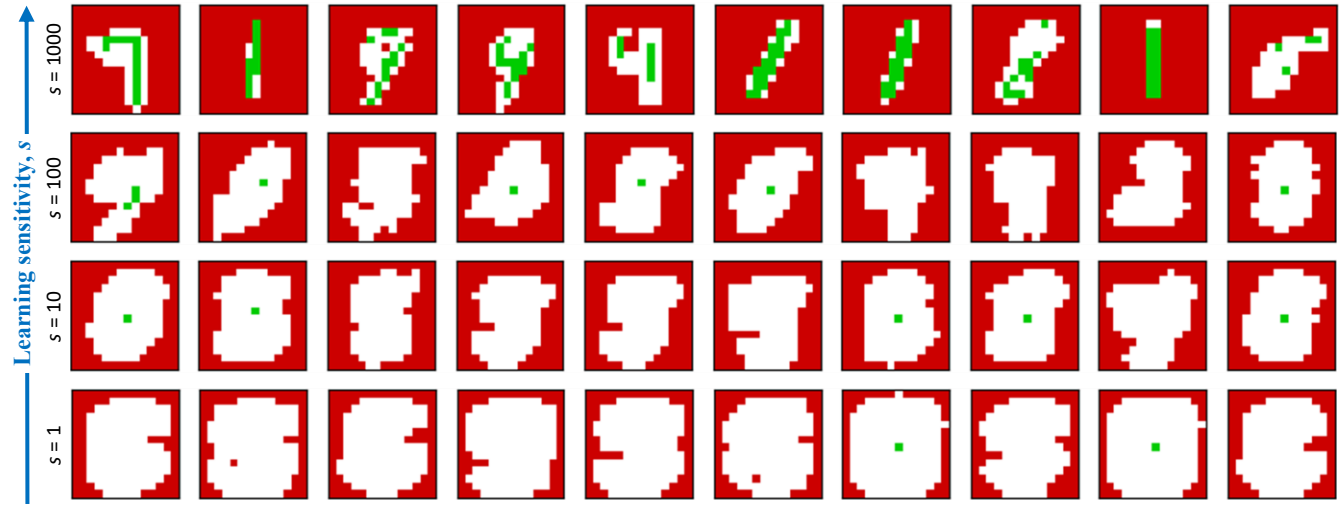

Fig. 8. Visualisation of TM (C=20, T=3, s=1..1000) *first-layer negative clauses* of Class "0" after training on the Digit MNIST dataset; each negative clause represents a certain template of classes other than Class "0"; the smaller the value of s, the wider the clause pattern.

Fig. 9. Visualisation of a *second-layer positive clause* of Class "0". The clause defines its class through: (i) negation of first-layer positive clauses of other classes; (ii) negation of first-layer negative clauses of Class "0"; (iii) reusing some negative clauses of other classes matching Class "0" samples; (iv) negation of some negative clauses of other classes corresponding to classes other than Class "0".

Instead, positive clauses of the second layer define a specific class by (see Fig. 9):

(i) negation of other classes (by ANDing negations of all first-layer positive clauses of other classes);

(ii) negation of first-layer negative clauses of the same class;

(iii) reusing some first-layer negative clauses from other classes that match the current clause class;

(iv) negation of some first-layer negative clauses of other classes that correspond to classes different from the current clause class.

For instance, a second-layer positive clause of Class "0" shown in Fig. 9 can be defined as:

$$Clause^{L2+}_{Class=0,j=0..C-1} = \\ = \neg Clause^{L1+}_{Class=1..9,j=1..C-1} \wedge \neg Clause^{L1-}_{Class=0,j=0..C-1} \wedge \\ \bigwedge_{i=0}^{k-1} \begin{pmatrix} (Clause^{L1-}_{Class=1..9,j=1..C} \vee \neg TA^{L2}_{i}) \wedge \\ (\neg Clause^{L1-}_{Class=1..9,j=1..C} \vee \neg TA^{L2}_{2k+i}) \end{pmatrix}.$$

In turn, negative clauses of the second layer specify data samples that do not belong to a specific class by (see Fig. 10):

(i) negation of first-layer positive clauses of the same class;

(ii) negation of some first-layer positive clauses of other classes;

(iii) negation of some first-layer negative clauses from other classes that match the current clause class.

At the same time, first-layer negative clauses of the same class will be excluded from negative clauses of the second layer. Following these reasonings, a second-layer negative clause for Class "0" shown in Fig. 10 can be specified as:

$$Clause^{L2-}_{Class=0,j=0..C-1} = \neg Clause^{L1+}_{Class=0,j=1..C-1} \wedge \\ \wedge_{i=0}^{k-1} \begin{pmatrix} (\neg Clause^{L1+}_{Class=1..9,j=1..C} \vee \neg TA^{L2}_{2k+i}) \wedge \\ (\neg Clause^{L1-}_{Class=1..9,j=1..C} \vee \neg TA^{L2}_{2k+i}) \end{pmatrix}.$$

Besides, after the first-layer TM has completed training, the state of some of its clauses may be suboptimal (i.e. some clauses may still be too specific or too broad, resulting in too few true predictions or too many false outputs). This can often happen if the value of the hyperparameter T is set too small. The second-layer TM has demonstrated the ability to detect such 'weak' clauses and eliminate them from the second-layer logical propositions.

CONCLUSION

This paper introduces a multi-layer architecture of Tsetlin Machines. It puts forward a hierarchical feature learning approach with the aim to further boost TM performance. It was shown that multi-layer TM can considerably outperform the single-layer architecture due to the accumulation and use of more information propagating from layer to layer still remaining a highly interpretable logic-based AI. For example, the second-layer clauses define the certain class via negation of first-layer positive clauses of other classes and reusing matching negative clauses.

The hierarchical training approach introduced in the paper also allows to reuse an array of Tsetlin Automata (main TM computation units) at different layers, thereby saving hardware resources and enabling distributed multi-layer learning.

It was also shown that while increasing the number of layers further improves training accuracy, it also causes overfitting issue similar to that seen in DNNs.

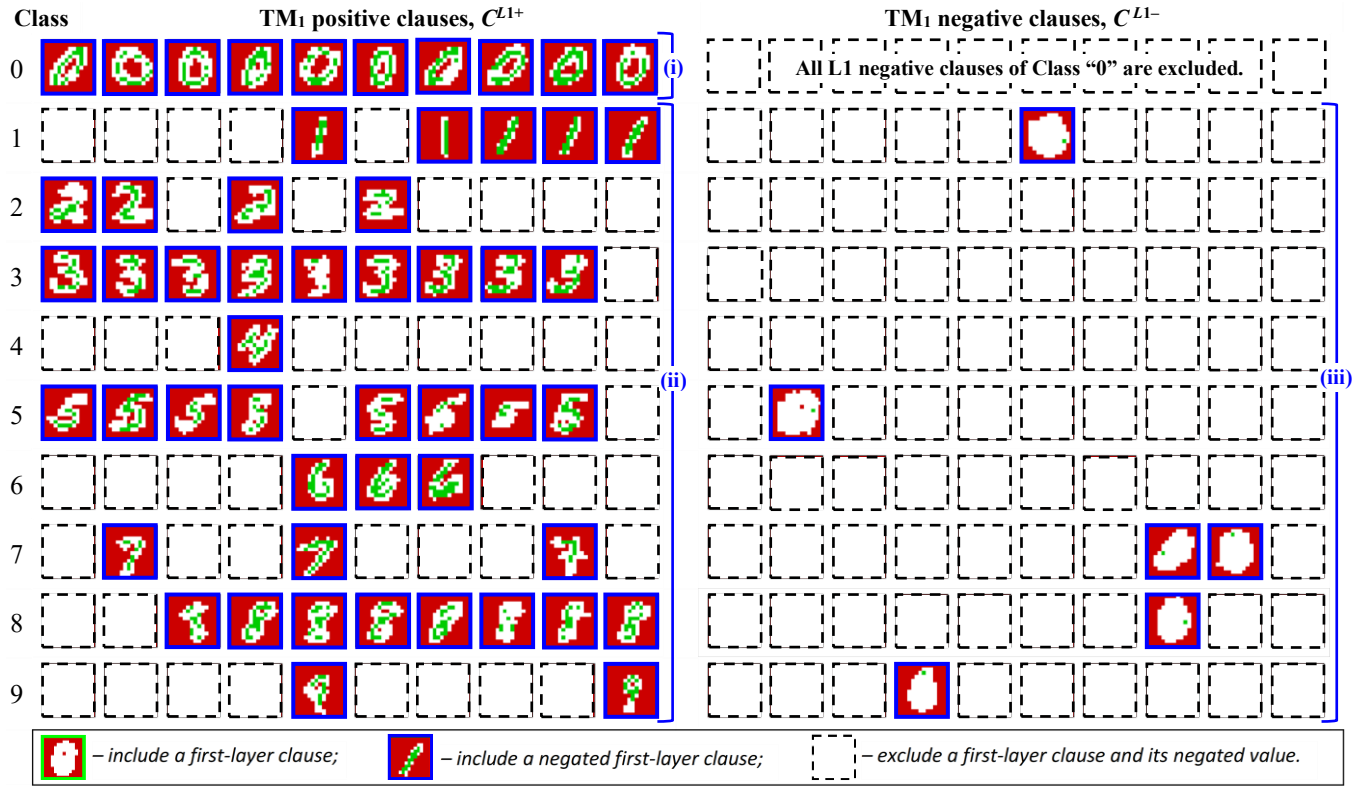

Fig. 10. Visualisation of a *second-layer negative clause* of Class "0". The clause defines its class through: (i) negation of first-layer positive clauses of Class "0"; (ii) negation of some first-layer positive clauses of other classes; (iii) negation of some first-layer negative clauses of other classes matching Class "0" samples.

Further research directions could be focused on developing more complex multi-layer TM architectures, including convolution layers. Besides, deep TM layers may benefit from the use of both conjunctive and disjunctive logic in the propositional clauses.

ACKNOWLEDGMENT

This research is supported by the British Academy's Researchers at Risk Fellowships Programme (RaR\100289).

REFERENCES

[1] O. Tarasyuk, A. Gorbenko, T. Rahman, R. Shafik and A. Yakovlev, "Logic-Based Machine Learning with Reproducible Decision Model Using the Tsetlin Machine," in *IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, Germany, 2023.

[2] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv,* vol. arXiv:1804.01508, pp. 1-14, 2018.

[3] M. L. Tsetlin, Automata Theory and Modeling of Biological Systems, New York: Academic Press, 1973, p. 288.

[4] V. Varshavsky and D. Pospelov, Puppets Without Strings: Reflections on the Evolution and Control of Some Man-Made Systems, Moscow: Mir, 1988, p. 294.

[5] K. Narendra and M. Thathachar, Learning Automata: An Introduction, Dover Publications, 2012, p. 496.

[6] S. A. Tunheim, L. Jiao, R. Shafik, A. Yakovlev and O.-C. Granmo, "A Convolutional Tsetlin Machine-based Field Programmable Gate Array Accelerator for Image Classification," in *International Symposium on the Tsetlin Machine (ISTM)*, 2022.

[7] S. Maheshwari, T. Rahman, R. Shafik, A. Yakovlev, A. Rafiev, L. Jiao and O.-C. Granmo, "REDRESS: Generating Compressed Models for Edge Inference Using Tsetlin Machines," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 45, no. 9, pp. 11152-11168, 2023.

[8] G. T. Berge, O.-C. Granmo, T. O. Tveit, M. Goodwin, L. Jiao and B. V. Matheussen, "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications," *IEEE Access,* vol. 7, pp. 115134-115146, 2019.

[9] K. D. Abeyrathna, O.-C. Granmo, X. Zhang and M. Goodwin, "Adaptive Continuous Feature Binarization for Tsetlin Machines Applied to Forecasting Dengue Incidences in the Philippines," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[10] J. Lei, T. Rahman, R. Shafik, A. Wheeldon, A. Yakovlev, O.-C. Granmo, F. Kawsar and A. Mathur, "Low-Power Audio Keyword Spotting Using Tsetlin Machines," *Journal of Low Power Electronics and Applications,* vol. 11, no. 2, pp. 1-18, 2021.

[11] K. D. Abeyrathna, H. S. G. Pussewalage, S. N. Ranasinghe, V. A. Oleshchuk and O.-C. Granmo, "Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[12] J. Lei, A. Wheeldon, R. Shafik, A. Yakovlev and O.-C. Granmo, "From Arithmetic to Logic based AI: A Comparative Analysis of Neural Networks and Tsetlin Machine," in *27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020.

[13] M. Kimura, T. Ouchi, H. Nakahama, K. Shima and S. Saito, "Learning automaton as a model to simulate and analyse learning behaviour in rats," *International Journal of Systems Science,* vol. 19, no. 10, pp. 2079-2090, 1988.

[14] T. Rahman, A. Wheeldon, R. Shafik, A. Yakovlev, J. Lei, O.-C. Granmo and S. Das, "Data Booleanization for Energy Efficient On-Chip Learning using Logic Driven AI," in *International Symposium on the Tsetlin Machine (ISTM)*, Grimstad, Norway, 2022.

[15] O. Tarasyuk, A. Gorbenko, T. Rahman, R. Shafik, A. Yakovlev, O.-C. Granmo and L. Jiao, "Systematic Search for Optimal Hyper-parameters of the Tsetlin Machine on MNIST Dataset," in *International Symposium on the Tsetlin Machine (ISTM)*, Newcastle-upon-Tyne, UK, 2023.

[16] L. Penrose, "The elementary statistics of majority voting," *Journal of the Royal Statistical Society,* vol. 109, no. 1, p. 53–57, 1947.

[17] K. Zyczkowski and W. Slomczynski, "Square Root Voting System, Optimal Threshold and π," in *Power, Voting, and Voting Power: 30 Years After*, M. Holler and H. Nurmi, Eds., Berlin, Heidelberg, Springer, 2013, p. 573–592.