

The Regression Tsetlin Machine: A Tsetlin Machine for Continuous Output Problems

K. Darshana Abeyrathna, Ole-Christoffer Granmo, Lei Jiao, and
Morten Goodwin

Centre for Artificial Intelligence Research, University of Agder, Grimstad, Norway
{darshana.abeyrathna, ole.granmo, lei.jiao, morten.goodwin}@uia.no

Abstract. The recently introduced Tsetlin Machine (TM) has provided competitive pattern classification accuracy in several benchmarks, composing patterns with easy-to-interpret conjunctive clauses in propositional logic. In this paper, we go beyond pattern classification by introducing a new type of TMs, namely, the *Regression Tsetlin Machine* (RTM). In all brevity, we modify the inner inference mechanism of the TM so that input patterns are transformed into a single continuous output, rather than to distinct categories. We achieve this by: (1) using the conjunctive clauses of the TM to capture arbitrarily complex patterns; (2) mapping these patterns to a continuous output through a novel voting and normalization mechanism; and (3) employing a feedback scheme that updates the TM clauses to minimize the regression error. The feedback scheme uses a new activation probability function that stabilizes the updating of clauses, while the overall system converges towards an accurate input-output mapping. The performance of the proposed approach is evaluated using six different artificial datasets with and without noise. The performance of the RTM is compared with the Classical Tsetlin Machine (CTM) and the Multiclass Tsetlin Machine (MTM). Our empirical results indicate that the RTM obtains the best training and testing results for both noisy and noise-free datasets, with a smaller number of clauses. This, in turn, translates to higher regression accuracy, using significantly less computational resources.

Keywords: Tsetlin Machine, Regression Tsetlin Machine, Tsetlin Automata, Regression Problems, Pattern Recognition, Propositional Logic.

1 Introduction

Computational simplicity, ease of interpretation, along with competitive pattern recognition accuracy, make the recently introduced Tsetlin Machine (TM) [1] a promising new paradigm for machine learning. Indeed, it has outperformed well-known machine learning algorithms such as Naïve Bayes, Logistic Regression, Neural Networks, and Support Vector Machine (SVM) in several benchmarks, including Iris Data Classification, Handwritten Digits Classification (MNIST), Predicting Optimum Moves in the Axis and Allies Board Game, and Classification of Noisy XOR Data with Non-Informative Features [1].

Tsetlin Automata and the Tsetlin Machine. The core of the TM is built on Tsetlin Automata (TAs), developed by M. L. Tsetlin in the early 1960s [2]. This powerful, yet simple, leaning mechanism has been used to solve a number of machine learning and stochastic optimization problems, such as resource allocation [3], stochastic searching on the line [4], distributed coordination [5], graph coloring [6], and forecasting disease outbreaks [7]. In the TM, TAs represent literals – input features and their negations. The literals, in turn, form conjunctive clauses in propositional logic, as decided by the TAs. The final TM output is a disjunction of all the specified clauses. In this manner, the pattern composition and learning procedure of the TM is fully transparent and understandable, facilitating human interpretation. In addition, the TM has an inherent computational advantage. That is, the inputs and outputs of the TM can naturally be represented as bits, and recognition and learning is performed by manipulating those bits. The operation of the TM thus demands relatively small computational resources, and supports hardware-near and parallel computation e.g. on GPUs.

Lately, the TM has provided state-of-the-art performance in several real-life applications. Berge et al. have for instance successfully used the TM for medical text categorization [8]. They used the TM to provide interpretable pattern recognition for the analysis of electronic health records. The authors demonstrated that the TM can outperform established machine learning algorithms such as k-nearest neighbors (kNN), SVM, Random Forest, Decision Trees, Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM) Neural Networks, and Convolutional Neural Networks (CNNs), in terms of precision, recall, and F-measure. Furthermore, Darshana et al. have shown that the TM can outperform MLPs, Decision Trees, and SVMs in dengue fever outbreak prediction. This result was achieved by making the TM capable of expressing thresholds and intervals that capture patterns formed by continuous features. By carefully selecting thresholds and intervals, the TM avoids losing information due to binarization [9].

Paper Contributions. In this paper, we introduce the Regression Tsetlin Machine (RTM) to overcome a rather severe limitation of the TM, namely, that it has been designed for classification, but not for continuous output. The RTM is a novel variant of the Classical Tsetlin Machine (CTM), specifically addressing the unique properties of regression. The novel modifications that we introduce are subtle, but crucial. First of all, the clause polarities the CTM uses to discriminate patterns, using positive and negative examples, are eliminated. Instead, the objective of the RTM is to use the clauses to map the sum of the clause outputs into one single continuous output. The discrepancy between predicted and target output is minimized with a new feedback scheme tailored for regression, including a modified stochastic activation probability function.

Paper Organization. The remainder of the paper is organized as follows. In Section 2, we present the main contribution of this paper, which is the RTM, and how we build it upon the CTM. We then investigate the behavior of the RTM using six different artificial datasets in Section 3. We demonstrate empirically

that the RTM is superior to CTM as well as its multi-class version when it comes to predicting continuous output. We conclude our work in Section 4.

2 The Regression Tsetlin Machine (RTM)

The RTM is a novel variant of the CTM. To highlight the unique properties of the RMT, we start this section with first reviewing the TM in more detail, and then discuss how it can be modified to support continuous output.

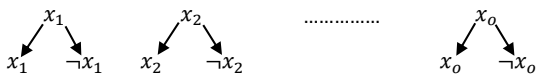
2.1 The Classical Tsetlin Machine (CTM)

At the heart of the TM, we find multiple teams of TAs that build conjunctive clauses in propositional logic. The purpose is to capture hidden patterns in the data.

Learning with TAs. Each Tsetlin Automaton (TA) learns the optimal action simply by interacting with its environment, identifying the action that provides the highest probability of reward [10, 11]. The TAs adjust their states based on the feedback they receive from the environment and eventually learn the best action. These simple learning devices are capable of online learning, have a simple structure, and require modest computational power. Yet, they are able to learn accurately with relatively few interactions with the environment [12, 13].

Clause Formation and the TA Team. The TM bases its operations on the simplest form of TAs, namely, the two action one, with finite memory depth. As illustrated in Table 1, a team of TAs cooperates to form a clause. The table depicts the steps leading to a clause being formed. Consider an input feature vector $\mathbf{X} = [x_1, x_2, \dots, x_o]$. Each TA represents either an input feature x_k or its negation $\neg x_k$ (referred to as a literal). Further, each TA in the team decides whether to include or exclude its assigned literal in the clause the team is forming. Accordingly, when there are o input features, $2 \times o$ TAs are needed to form the clause. The two actions available to each TA are $\{in, ex\}$. Here, *in* refers to including the literal controlled by the TA and *ex* refers to excluding it.

Table 1. The steps used to form a clause based on the input features and the actions of the TAs.

Phase	Operations								Comments
1									Input Features, \mathbf{X}
2	x_1	$\neg x_1$	x_2	$\neg x_2$	x_o	$\neg x_o$		Literals
3	TA_1^1	TA_1^2	TA_2^1	TA_2^2	TA_o^1	TA_o^2		TA
4	<i>in</i>	<i>ex</i>	<i>ex</i>	<i>in</i>	<i>ex</i>	<i>in</i>		Actions = $\{in, ex\}$
5	$x_1 \wedge \neg x_2 \wedge \dots \wedge \neg x_o$								$c_1 = \bigwedge_{k=1}^o x_k, \quad \forall TA_k^1 = in$ $c_2 = \bigwedge_{k=1}^o \neg x_k, \quad \forall TA_k^2 = in$ $C = c_1 \wedge c_2$

As seen in the final step in the table, the included literals form a conjunctive clause, while the excluded ones are ignored.

Clauses and Voting. The number of clauses, m , needed for a particular problem depends on the complexity of the dataset. It should at least be sufficient to cover the full range of sub-patterns associated with each output $\{0, 1\}$. However, with hidden and unknown sub-patterns, a grid search is required to find the best m .

The m clauses are assigned either a positive or negative polarity, and they vote separately to decide the final output of the TM. Clauses with odd index are assigned positive polarity (C^+) and they vote for the final output 1. Clauses with even index are assigned negative polarity (C^-) and they vote for the final output 0. For both categories, a vote is submitted when the clause recognizes a sub-pattern. If the clause is unable to find a sub-pattern, it declines to vote. Finally, the output, y , is decided based on the number of votes gained by each category $\{0, 1\}$ as given in the Eq. (1):

$$y = \begin{cases} 1, & \text{if } \sum_{j=1,3,m-1} C_j^+ > \sum_{j=2,4,m} C_j^- \\ 0, & \text{if } \sum_{j=1,3,m-1} C_j^+ < \sum_{j=2,4,m} C_j^- \end{cases} \quad (1)$$

Learning Procedure. Learning in the TM is based on reinforcement learning. The reward, penalty, and inaction probabilities that guide the TAs in all of the clauses depend on several factors, namely, the actual output, clause output, literal value, and current state of the TA. The basic idea is to alter the number of votes belong to each output category when the output is a false negative or a false positive. In the TM, this is done by two types of feedback – Type I and Type II. Type I feedback eliminates false negative output, while Type II feedback eliminates false positive output. Both of these kinds of feedback are summarized in Table 2.

Type I feedback is given to clauses with positive polarity when the actual output, \hat{y} , is 1 and clauses with negative polarity when the actual output, \hat{y} , is 0. The probability of activation of Type I feedback is $[T - \max(-T, \min(T, \sum_{j=1}^m C_j))]/2T$. Type II feedback is given to clauses with positive polarity when the actual output, \hat{y} , is 0 and clauses with negative polarity when the actual output, \hat{y} , is 1. The

Table 2. Type I and Type II feedback designed to eliminate false negative and false positive output.

Feedback Type			I				II			
Clause Output			1		0		1		0	
Literal Value			1	0	1	0	1	0	1	0
Current State	Include	Reward Probability	(s-1)/s	NA	0	0	0	NA	0	0
		Inaction Probability	1/s	NA	(s-1)/s	(s-1)/s	1	NA	1	1
		Penalty Probability	0	NA	1/s	1/s	0	NA	0	0
	Exclude	Reward Probability	0	1/s	1/s	1/s	0	0	0	0
		Inaction Probability	1/s	(s-1)/s	(s-1)/s	(s-1)/s	1	0	1	1
		Penalty Probability	(s-1)/s	0	0	0	0	1	0	0

† s is the precision and controls the granularity of the sub-patterns captured [1]

probability of activation of Type II feedback is $[T + \max(-T, \min(T, \sum_{j=1}^m C_j))]/2T$. TAs remain unchanged if the vote difference, $\sum_{j=1}^m C_j$, is higher than or equal to T when $\hat{y} = 1$ and lower than or equal to $-T$ when $\hat{y} = 0$, according to the activation probabilities of each type of feedback.

In all brevity, when the actual output is 1, ($\hat{y} = 1$), the votes from clauses with positive polarity must outnumber the votes from clauses with negative polarity. Therefore, clauses with positive polarity receive Type I feedback (the activation probability increases with the number of voting clauses with negative polarity) since it reinforces clauses which output 1. Similarly, clauses with negative polarity receive Type II feedback (the activation probability increases with the number of voting clauses with positive polarity) since it suppresses voting activity by making clauses of negative polarity evaluate to 0. The procedure is similar when the actual output is 0, ($\hat{y} = 0$). The TM then needs to make sure that more clauses with negative polarity provide votes compared to those with positive polarity. Eventually, this feedback reduces the number of false positives and false negatives to make the TM to learn the propositional formula that can apply on unseen data to provide high accuracy output.

2.2 The Multi-Class Tsetlin Machine (MTM)

The summation operator of the CTM at the end aggregates all clause outputs into one of the two outputs, 0 or 1. However, for categorization tasks with more classes than two, another design is needed – the Multi-Class Tsetlin Machine (MTM). In the MTM, clauses are partitioned equally among the classes. The clauses of each individual class then act separately, similarly to a single TM. However, the votes output for each class then form the basis for classification. That is, an argmax operator arbitrates the final class, based on the votes collected for each class. When there are n classes, the output, y , can thus be expressed as:

$$y = \operatorname{argmax}_{i=1, \dots, n} \left\{ \left(\sum_{j=1, 3, \dots, (\frac{m}{n})-1} C_j^i - \sum_{j=2, 4, \dots, (\frac{m}{n})} C_j^i \right) \right\}. \quad (2)$$

The training procedure is similar to the CTM training procedure. However, in the MTM, the clauses of the class being the target of the current training sample are treated as if $\hat{y} = 1$, while the clauses of a randomly selected class from the remaining classes is treated as if $\hat{y} = 0$. In each class, clauses with positive polarity vote to say that the output belongs to the considered class. Similarly, the clauses with negative polarity vote to indicate that the output does not belong to the considered class.

2.3 The Regression Tsetlin Machine (RTM)

When the output is continuous, neither the CTM or the MTM above are ideal. However, we will now show that the CTM can be modified to produce continuous output by means of three pertinent modifications.

In CTM and MTM, the polarity of clauses was used to classify data into different classes. We now remove the polarity of clauses, since we intend to use the clauses as additive building blocks that can be used to calculate continuous output. That is, we intend to map the total vote count into a continuous output. As a result, the complexity of the RTM is actually reduced.

With merely one type of clause, the summation operator outputs a value between 0 and T , which is simply the number of clauses that evaluated to 1. This value is then normalized to produce the regression output. Thus, through this simple modification, the TM can now produce continuous output, with precision that increases with higher T .

Let \hat{y}_{\max} denote the maximum output value \hat{y} among the N training samples $\mathbf{Y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_N]$. Then the sum of the votes from the clauses $\sum_{j=1}^m C_j$ of the TM is normalized to achieve the regression output by dividing by T and multiplying with \hat{y}_{\max} . So, for the o^{th} training sample, (\hat{X}_o, \hat{y}_o) , the TM output, y_o , is calculated from the input \hat{X}_o as follows:

$$y_o = \frac{\sum_{j=1}^m C_j(\hat{X}_o) \times \hat{y}_{\max}}{T} . \quad (3)$$

Feedback, then, is based on comparing the output, y_o of the TM with the target output \hat{y}_o . The target value \hat{y}_o can be higher or lower than the output value y_o . This is our basis for our new feedback scheme. That is, similarly to other machine learning methods, certain internal operations are needed to minimize the error between the predicted output, y_o , and target output, \hat{y}_o . In the RTM, this is quite simply achieved by providing Type I and Type II feedbacks according to the following criteria:

$$Feedback = \begin{cases} \text{Type I,} & \text{if } y_o < \hat{y}_o , \\ \text{Type II,} & \text{if } y_o > \hat{y}_o . \end{cases} \quad (4)$$

Similar to the CTM, the idea here is to increase the number of clauses that output 1 when the predicted output is less than the target output ($y_o < \hat{y}_o$). To achieve this, we then provide Type I feedback. Conversely, Type II feedback is applied to decrease the number of clauses that evaluate to 1 when the predicted output is higher than the target output ($y_o > \hat{y}_o$).

To stabilize learning, however, we use an activation probability function that makes the probability of giving a clause feedback proportional to the difference between the predicted and target output (the error). That is, in the RTM, feedback to clauses is determined stochastically using the activation probability function, P_{act} :

$$P_{act} = \frac{K \times |y_o - \hat{y}_o|}{\hat{y}_{\max}} . \quad (5)$$

The probability can be adjusted according to the number of clauses using the activation probability adjuster, K , in the above activation function. The

activation function together with K , reduces the oscillation of the change of the predicted value during the training process and stabilizes it around the target value. The resulting behavior of the RTM is studied in the following sections and compared against the CTM and MTM.

3 Empirical Results

3.1 Experiment Setup

The behavior of the RTM is studied using six different datasets. Dataset I contains 2-bit feature input and the output is 100 times larger than the decimal value of the binary input (e.g., when the input is $[1, 0]$, the output is 200). The training set consists of 8000 samples while testing set consists of 2000 samples, both without noise. Dataset II contains the same data as Dataset I, except that the output of the training data is perturbed to introduce noise. For Dataset III we introduce 3-bit input, without noise, and for Dataset IV we have 3-bit input with noisy output. Dataset V has 4-bit input without noise, and Dataset VI has 4-bit input with noisy.

The input feature vectors have been generated with equal probability of 0 and 1 values, leading to a more or less uniform distribution of bit values.

In order to increase our understanding of the Regression Tsetlin Machine, we also studied the effect the hyper-parameters T and s have on learning.

Experiment I: We first studied the effect varying T had on performance for the different datasets.

Experiment II: The effect of different s values (controlling the number of sub-patterns) was further investigated on all of the datasets.

Experiment III: We finally compare the RTM results with what can be achieved with TM and MTM.

3.2 Results and Discussion

We use Mean Absolute Error (MAE) to measure performance. Fig. 1 plots error across 200 epochs, with learning influenced by different T . Fig. 1(a) shows the results for Dataset I, and Fig. 1(b) reports results for Dataset II, and so on. MAE after 200 epochs is also given in brackets for each threshold in the legend.

From Fig. 1, we can observe that just 3 clauses ($T = 3$) are enough to reduce error to zero for Dataset I, which can be explained by the noise-free data. Because the output value is decided by the number of clauses that output 1, we require *two* clauses with $TA_1^1 = \{in\}$, $TA_1^2 = \{ex\}$, $TA_2^1 = \{ex\}$, and $TA_2^2 = \{ex\}$ in Phase 4 in Table 1 to capture the pattern $(1 \ast)$ and *one* clause with $TA_1^1 = \{ex\}$, $TA_1^2 = \{ex\}$, $TA_2^1 = \{in\}$, and $TA_2^2 = \{ex\}$ in Phase 4 of Table 1 to capture the pattern $(\ast 1)$. Here, \ast means an input feature that can take an arbitrary value, either 0 or 1. These three clauses can collectively form any outputs for

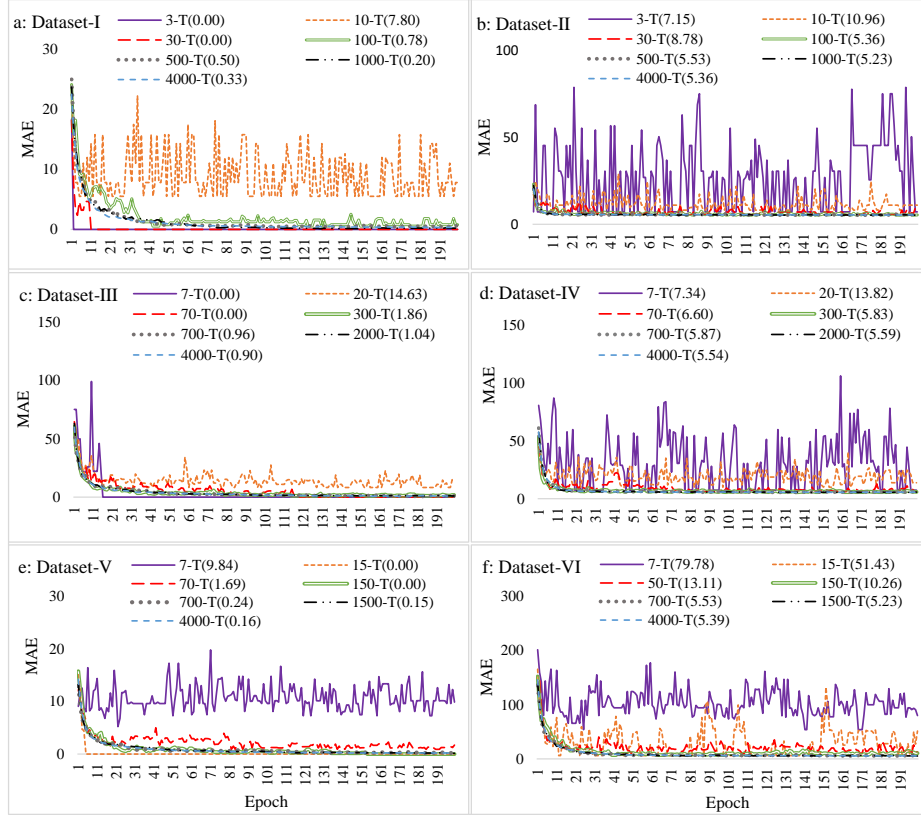


Fig. 1. Training error over training epochs. Each dataset is processed with different T .

the Dataset I as shown in Table 3. For instance, input (0 1) only activates the clause with $TA_1^1 = \{ex\}$, $TA_1^2 = \{ex\}$, $TA_2^1 = \{in\}$, and $TA_2^2 = \{ex\}$ to represent the pattern $(* 1)$ and correctly computes the output, 100. Likewise, input (1 0) only activates the *two* clauses with $TA_1^1 = \{in\}$, $TA_1^2 = \{ex\}$, $TA_2^1 = \{ex\}$, and $TA_2^2 = \{ex\}$ to represent the pattern $(1 *)$, and correctly computes the output, 200. All the clauses are activated when the input is (1 1) and therefore the output 300 is correctly computed.

We observe similar behaviour for Dataset III and Dataset V. More specifically, Dataset III requires *seven* clauses to represent the three different patterns it contains, namely, $(4 \times (1 **), 2 \times (* 1 *), 1 \times (** 1))$ ¹ and Dataset V requires *fifteen* clauses to represent four different patterns in it $(8 \times (1 ** *), 4 \times (* 1 * *), 2 \times (** 1 *), 1 \times (** * 1))$. As we can see from these 3 datasets, RTM can reach 0.00 for the training MAE when T is a multiplier of the minimum required clauses. For example, Dataset I can be perfectly learned when there are 30 clauses.

¹ In this expression, “*four* clauses to represent the pattern $(1 ** *)$ ” is written as “ $4 \times (1 ** *)$ ”

Table 3. Computing output for different datasets by activating different clauses.

Dataset	Output	Required number of clauses to represent different patterns ^{††}
I	0	None
	100	$1 \times (* 1)$
	200	$2 \times (1 *)$
	300	$2 \times (1 *) + 1 \times (* 1)$
III	0	None
	100	$1 \times (* * 1)$
	200	$2 \times (* 1 *)$
	300	$2 \times (* 1 *) + 1 \times (* * 1)$
	400	$4 \times (1 * *)$
	500	$4 \times (1 * *) + 1 \times (* * 1)$
	600	$4 \times (1 * *) + 2 \times (* 1 *)$
	700	$4 \times (1 * *) + 2 \times (* 1 *) + 1 \times (* * 1)$
V	0	None
	100	$1 \times (* * * 1)$
	200	$2 \times (* * 1 *)$
	300	$2 \times (* * 1 *) + 1 \times (* * * 1)$
	400	$4 \times (* 1 * *)$
	500	$4 \times (* 1 * *) + 1 \times (* * * 1)$
	600	$4 \times (* 1 * *) + 2 \times (* * 1 *)$
	700	$4 \times (* 1 * *) + 2 \times (* * 1 *) + 1 \times (* * * 1)$
	800	$8 \times (1 * * *)$
	900	$8 \times (1 * * *) + 1 \times (* * * 1)$
	1000	$8 \times (1 * * *) + 2 \times (* * 1 *)$
	1100	$8 \times (1 * * *) + 2 \times (* * 1 *) + 1 \times (* * * 1)$
	1200	$8 \times (1 * * *) + 4 \times (* 1 * *)$
	1300	$8 \times (1 * * *) + 4 \times (* 1 * *) + 1 \times (* * * 1)$
	1400	$8 \times (1 * * *) + 4 \times (* 1 * *) + 2 \times (* * 1 *)$
	1500	$8 \times (1 * * *) + 4 \times (* 1 * *) + 2 \times (* * 1 *) + 1 \times (* * * 1)$

^{††} for example, “two clauses to represent the pattern (1 *)” is written as “ $2 \times (1 *)$ ”

However, when T is not a multiplier of the minimum required clauses, RTM cannot align its output, y_o , to the target output, \hat{y}_o , during the training phase. For instance, when there are *four* clauses to work with Dataset I, the training will end up by having *three* clauses to represent the pattern (1 *) or *two* clauses to represent the pattern (* 1). This combination then cannot output one or more values in the target. For example, if there are *three* clauses for the pattern (1 *) and *one* clause for the pattern (* 1) after training, input (1 0) activates the clauses that represent the pattern (1 *), resulting an incorrect output that is 300. Likewise, input (1 1) activates all four clauses to incorrectly compute the output which is 400.

As a strategy for problems where the number of clauses is unknown and in order to work with real applications where noise plays a significant role, the RTM can start with a large T . Then, since the output, y_o , is a fraction of the threshold, T , the error decreases. This behaviour is verified empirically in Fig. 1, showing how increasing T leads to reduced error.

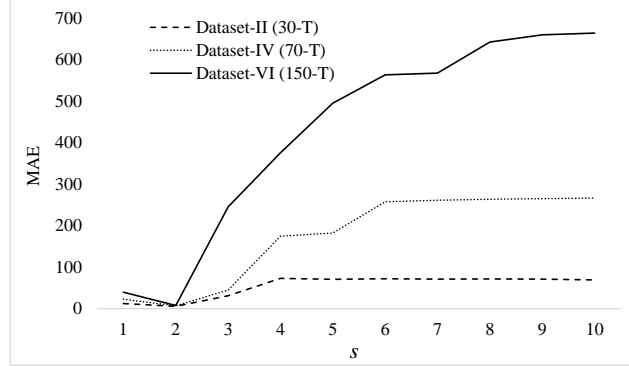


Fig. 2. Variation of MAE over different s for fixed T .

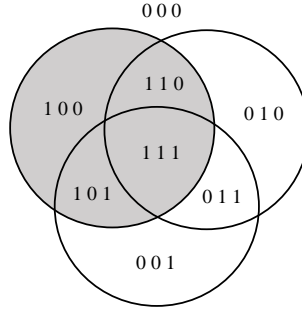


Fig. 3. Pattern distribution for the 3-bits input datasets.

The effect of s is studied by increasing it from 1.0 to 10.0 for Dataset II, Dataset IV, and Dataset VI, with fixed T . Fig. 2 shows the variation of MAE over various s values for noisy data. The MAE decreases when s increases from 1.0 to 2.0. The MAE becomes larger when s increases from 2.0, and then becomes stable afterwards. However, for all considered datasets, the optimum s , where the RTM learns all datasets with minimum MAE , is equal to 2.0. The reason can be explained with the aid of Fig. 3, where one sees the distribution of patterns when the dataset has 3 input bits.

The probability of any of these patterns is $1/8$ since there are overall 8 unique patterns. However, to capture the pattern $(1 * *)$ (shaded area), according to the TM dynamics [1], $1/s$ should be equal to the probability of the considered pattern, which is $4/8 (=1/2)$ and hence $s = 2$. For instance, if someone assigns $s = 4$, clauses will start to learn much finer patterns, such as $(1 0 *)$, $(1 1 *)$, and $(0 1 *)$. This significantly increases the number of clauses needed to capture the sub-patterns and unstabilize learning. This is also the case for Dataset II and Dataset VI, with the probability that $(1 *)$ occurs, i.e., $2/4 (=1/2)$ and that $(1 * * *)$ occurs, i.e., $7/14 (=1/2)$, respectively.

To compare the performance of the RTM with CTM and MTM, each model is tested with different T values. The training and testing MAE for all the cases are summarized in Table 4 and 5, respectively.

Table 4. Training *MAE* after 200 training epochs with different *T* on various methods.

		<i>T</i>	RTM							CTM		MTM		
			3	10	30	100	500	1000	4000	6	8000	1000	10000	16000
			MAE											
Dataset	1	<i>MAE</i>	0.0	7.8	0.0	0.8	0.5	0.2	0.3	7.7	0.0	0.0	0.0	0.0
	2	<i>MAE</i>	7.2	11.0	8.8	5.4	5.5	5.2	5.4	11.1	24.1	8.4	7.1	7.9
		<i>T</i>	7	20	70	300	700	2000	5000	14	8000	2000	10000	16000
	3	<i>MAE</i>	0.0	14.6	0.0	1.9	1.00	1.0	0.9	0.0	0.0	18.3	0.0	0.0
	4	<i>MAE</i>	7.4	13.8	6.6	5.8	5.9	5.6	5.5	111.3	13.3	14.2	8.8	8.4
		<i>T</i>	7	15	70	150	700	1500	4000	30	8000	4000	10000	16000
	5	<i>MAE</i>	9.8	0.0	1.7	0.0	0.2	0.2	0.2	149.7	158.7	373.1	0.0	0.0
	6	<i>MAE</i>	79.8	51.4	13.1	10.3	5.5	5.3	5.4	181.5	96.4	449.9	8.0	7.8

Table 5. Testing *MAE* for different *T* on various methods.

		<i>T</i>	RTM							CTM		MTM		
			3	10	30	100	500	1000	4000	6	8000	1000	10000	16000
			MAE											
Dataset	1	<i>MAE</i>	0.0	7.6	0.0	0.8	0.5	0.2	0.3	9.0	0.0	0.0	0.0	0.0
	2	<i>MAE</i>	5.0	10.6	7.1	1.2	2.7	1.6	1.8	9.4	25.3	7.5	5.4	7.0
		<i>T</i>	7	20	70	300	700	2000	5000	14	8000	2000	10000	16000
	3	<i>MAE</i>	0.0	14.2	0.0	2.1	1.0	1.2	1.0	0.0	0.0	22.0	0.0	0.0
	4	<i>MAE</i>	5.0	14.5	4.2	3.3	3.4	1.9	2.7	98.5	12.5	16.0	8.7	8.3
		<i>T</i>	7	15	70	150	700	1500	4000	30	8000	4000	10000	16000
	5	<i>MAE</i>	9.9	0.0	1.8	0.0	0.3	0.2	0.2	154.6	155.5	372.9	0.0	0.0
	6	<i>MAE</i>	78.0	50.1	12.5	8.5	3.5	2.7	2.8	191.3	102.4	431.3	6.9	6.7

The training and testing *MAE* values reach zero when the RTM operates with noise free data and when *T* equals to the optimum clauses required or its multipliers. When the optimum *T* is unknown, and when data is noisy, applying a higher *T* is beneficial. As an example, Dataset III, which has 3 bits as inputs, can be perfectly learned with *T* equal to 7 and 70. For the same dataset, RTM acquires training *MAE* 1.0 with *T* equaling to 700, which is better than *MAE* (14.2) when *T* equals to 20.

The outputs are converted to bits and each bit position is then trained and predicted separately in CTM. A fair comparison is conducted between these two approaches by assigning the best *T* from RTM to CTM as the number of clauses (the number of clauses for CTM = the best *T* of RTM \times 2, since CTM has clauses with positive and negative polarities). According to the training and testing *MAE*, CTM works better with less complex datasets such as Dataset I and Dataset III. However, with a higher number of inputs and with the noise involved in the training data, the performance decreases.

MTM requires a large number of clauses by nature when it works with continuous outputs since it has to consider all possible values from 0 to \hat{y}_{\max} as distinct classes (e.g. 300 classes for Dataset I and Dataset II, 700 classes for Dataset III and Dataset IV). According to the training and testing *MAE* in the above tables, MTM requires roughly 3 clauses or more per class. For instance, the features in Dataset I can be learned with 1000 clauses yet that amount is

insufficient to work with Dataset III and Dataset V. However, all these noise free datasets can be learned with 10000 or more clauses, but the accuracy is accompanied with a large computational power as price.

As a summary, RTM obtains the best training and testing *MAE* for both noisy and noise free data with a smaller number of clauses compared with the CTM and MTM. According to the test results of Dataset II, Dataset IV, and Dataset VI, which are more relevant to the results in real applications, the minimum *MAE* values obtained by RTM for these three Datasets are 1.2, 1.9, and 2.7, respectively. The average of these minimum *MAE* values (1.93) is approximately 20 and 3.5 times lower compared with the average of minimum *MAE* values obtained by CTM and MTM for the same datasets, respectively. In terms of the number of clauses required to achieve the above results, RTM utilizes 1000 clauses to obtain the highest of these three minimum *MAE*, while CTM and MTM utilize 8 and 16 times more clauses, respectively. This difference is characteristic for RTM — it provides better *MAE* with less computational power.

4 Conclusion

In this paper we proposed the Regression Tsetlin Machine (RTM), a novel variant of the Classic Tsetlin Machine that supports continuous output in regression problems. In RTM, the polarities in clauses were removed and the total clause output was normalized to produce continuous output predictions. The number of affected clauses to receive the feedback in RTM was decided stochastically using a linear activation probability function. The prediction power of this novel approach was studied using six different datasets, with noise free and noisy training data. Our empirical results show significantly better performance of RTM compared with CTM and MTM, both in terms of training and the testing error, as well as the computational power required.

References

1. O.-C. Granmo, “The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic,” *arXiv:1804.01508*.
2. M. L. Tsetlin, “On Behaviour of Finite Automata in Random Medium,” *Avtom I Telemekhanika*, vol. 22, pp. 1345–1354, 1961.
3. O.-C. Granmo and B. J. Oommen, “Solving Stochastic Nonlinear Resource Allocation Problems Using a Hierarchy of Twofold Resource Allocation Automata,” *IEEE Transaction on Computers*, 2010.
4. B. J. Oommen, S.-W. Kim, M. T. Samuel, and O.-C. Granmo, “A Solution to the Stochastic Point Location Problem in Metalevel Nonstationary Environments,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 2, pp. 466–476, 2008.
5. B. Tung and L. Kleinrock, “Using Finite State Automata to Produce Self-Optimization and Self-Control,” *IEEE transactions on parallel and distributed systems*, vol. 7, no. 4, pp. 439–448, 1996.

6. N. Bouhmala and O.-C. Granmo, "Stochastic Learning for SAT-Encoded Graph Coloring Problems," *International Journal of Applied Metaheuristic Computing(IJAMC)*, vol. 1, no. 3, pp. 1–19, 2010.
7. K. D. Abeyrathna, O.-C. Granmo, and M. Goodwin, "A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines," *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 680–685, IEEE, 2018.
8. G. T. Berge, O.-C. Granmo, T. Oddbjørn Tveit, M. Goodwin, L. Jiao, and B. Viggo Matheussen, "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization with Medical Applications," *arXiv e-prints*, p. *arXiv:1809.04547*, Sep 2018.
9. K. D. Abeyrathna, O.-C. Granmo, X. Zhang, and M. Goodwin, "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks," *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 2019.
10. S. Misra, P. V. Krishna, and K. I. Abraham, "Simple Learning Automata-Based Solution for Intrusion Detection in Wireless Sensor Networks," *Wireless Communications and Mobile Computing*, vol. 11, no. 3, pp. 426–441, 2011.
11. T. A. Tuan, L. C. Tong, and A. Premkumar, "An Adaptive Learning Automata Algorithm for Channel Selection in Cognitive Radio Network," *2010 International Conference on Communications and Mobile Computing*, Courier Corporation, 2012.
12. K. S. Narendra and M. A. Thathachar, "Learning Automata: An Introduction," *2010 International Conference on Communications and Mobile Computing*, pp. 159–163, IEEE, 2010.
13. K. Narendra and M. Thathachar, "Learning Automata-A Survey," *IEEE Transactions on systems, man, and cybernetics*, no. 4, pp. 323–334, 1974.