Integer Weighted Regression Tsetlin Machines

Kuruge Darshana Abeyrathna(✉), Ole-Christoffer Granmo,
and Morten Goodwin

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

Abstract. The Regression Tsetlin Machine (RTM) addresses the lack of interpretability impeding state-of-the-art nonlinear regression models. It does this by using conjunctive clauses in propositional logic to capture the underlying non-linear frequent patterns in the data. These, in turn, are combined into a continuous output through summation, akin to a linear regression function, however, with non-linear components and binary weights. However, the resolution of the RTM output is proportional to the number of clauses employed. This means that computation cost increases with resolution. To address this problem, we here introduce integer weighted RTM clauses. Our integer weighted clause is a compact representation of multiple clauses that capture the same sub-pattern—w repeating clauses are turned into one, with an integer weight w. This reduces computation cost w times, and increases interpretability through a sparser representation. We introduce a novel learning scheme, based on so-called stochastic searching on the line. We evaluate the potential of the integer weighted RTM empirically using two artificial datasets. The results show that the integer weighted RTM is able to acquire on par or better accuracy using significantly less computational resources compared to regular RTM and an RTM with real-valued weights.

Keywords: Tsetlin machines · Regression tsetlin machines · Weighted tsetlin machines · Interpretable machine learning · Stochastic searching on the line

1 Introduction

The recently introduced Regression Tsetlin Machine (RTM) [1,2] is a propositional logic based approach to interpretable non-linear regression, founded on the Tsetlin Machine (TM) [3]. Being based on disjunctive normal form, like Karnaugh maps, the TM can map an exponential number of input feature value combinations to an appropriate output [4]. Recent research reports several distinct TM properties. The clauses that a TM produces have an interpretable form (e.g., **if** X **satisfies** condition A **and not** condition B **then** Y = 1), similar to the branches in a decision tree [5]. For small-scale pattern recognition

H. Fujita et al. (Eds.): IEA/AIE 2020, LNAI 12144, pp. 686–694, 2020.
https://doi.org/10.1007/978-3-030-55789-8_59

problems, up to three orders of magnitude lower energy consumption and inference time has been reported, compared to neural networks alike [6]. Like neural networks, the TM can be used in convolution, providing competitive memory usage, computation speed, and accuracy on MNIST, F-MNIST and K-MNIST, in comparison with simple 4-layer CNNs, K-Nereast Neighbors, SVMs, Random Forests, Gradient Boosting, BinaryConnect, Logistic Circuits, and ResNet [7]. By introducing clause weights that allow one clause to represent multiple, it has been demonstrated that the number of clauses can be reduced up to 50×, without loss of accuracy, leading to more compact clause sets [4]. Finally, hyper-parameter search can be simplified with multi-granular clauses, eliminating the pattern specificity parameter [8].

Paper Contributions: In the RTM, regression resolution is proportional to the number of conjunctive clauses employed. In other words, computation cost and memory usage grows proportionally with resolution. Building upon the Weighted TM (WTM) by Phoulady et al. [4], this paper introduces weights to the RTM scheme. However, while the WTM uses real-valued weights for classification, we instead propose a novel scheme based on *integer* weights, targeting *regression.* In brief, we use the stochastic searching on the line approach pioneered by Oommen in 1997 [9] to eliminate multiplication from the weight updating. In addition to the computational benefits this entails, we also argue that integer weighted clauses are more interpretable than real-valued ones because they can be seen as multiple copies of the same clause. Finally, our scheme does not introduce additional hyper-parameters, whereas the WTM relies on weight learning speed.

Paper Organization: The remainder of the paper is organized as follows. In Sect. 2, the basics of RTMs are provided. Then, in Sect. 3, the SPL problem and its solution are explained. The main contribution of this paper, the integer weighting scheme for the RTM, is presented in detail in Sect. 4 and evaluated empirically using two artificial datasets in Sect. 5. We conclude our work in Sect. 6.

2 The Regression Tsetlin Machine (RTM)

The RTM performs regression based on formulas in propositional logic. In all brevity, the input to an RTM is a vector $\mathbf{X}$ of o propositional variables x_k, $\mathbf{X} \in \{0,1\}^o$. These are further augmented with their negated counterparts $\bar{x}_k = 1 - x_k$ to form a vector of literals: $\mathbf{L} = [x_1, \ldots, x_o, \bar{x}_1, \ldots, \bar{x}_o] = [l_1, \ldots, l_{2o}]$. In contrast to a regular TM, the output of an RTM is real-valued, normalized to the domain $y \in [0,1]$.

Regression Function: The regression function of an RTM is simply a linear summation of products, where the products are built from the literals:

$$y = \frac{1}{T} \sum_{j=1}^{m} \prod_{k \in I_j} l_k. \tag{1}$$

Above, the index j refers to one particular product of literals, defined by the subset I_j of literal indexes. If we e.g. have two propositional variables x_1 and x_2, the literal index sets $I_1 = \{1, 4\}$ and $I_2 = \{2, 3\}$ define the function: $y = \frac{1}{T}(x_1 \bar{x}_2 + \bar{x}_1 x_2)$. The user set parameter T decides the resolution of the regression function. Notice that each product in the summation either evaluates to 0 or 1. This means that a larger T requires more literal products to reach a particular value y. Thus, increasing T makes the regression function increasingly fine-grained. In the following, we will formulate and refer to the products as *conjunctive clauses*, as is typical for the regular TM. The value c_j of each product is then a conjunction of literals:

$$c_j = \prod_{k \in I_j} l_k = \bigwedge_{k \in I_j} l_k. \tag{2}$$

Finally, note that the number of conjunctive clauses m in the regression function also is a user set parameter, which decides the expression power of the RTM.

Tsetlin Automata Teams: The composition of each clause is decided by a team of Tsetlin Automata (TAs) [10]. There are $2 \times o$ number of TAs per clause j. Each represents a particular literal k and decides whether to *include* or *exclude* that literal in the clause. The decision depends on the current state of the TA, denoted $a_{j,k} \in \{1, \ldots, 2N\}$. States from 1 to N produce an *exclude* action and states from $N+1$ to $2N$ produce an *include* action. Accordingly, the set of indexes I_j can be defined as $I_j = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$. The states of all of the TAs are organized as an $m \times 2o$ matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$ where m is the number of clauses.

Learning Procedure: Learning in RTM is done through an online reinforcement scheme that updates the state matrix $\mathbf{A}$ by processing one training example $(\hat{X}_i, \hat{y}_i)$ at a time, as detailed below.

The RTM employs two kinds of feedback, Type I and Type II, further defined below. Type I feedback triggers TA state changes that eventually make a clause output 1 for the given training example $\hat{X}_i$. Conversely, Type II feedback triggers state changes that eventually make the clause output 0. Thus, overall, regression error can be systematically reduced by carefully distributing Type I and Type II feedback:

$$Feedback = \begin{cases} \text{Type I}, & \text{if} \quad y < \hat{y}_i, \\ \text{Type II}, & \text{if} \quad y > \hat{y}_i. \end{cases} \tag{3}$$

In effect, the number of clauses that evaluates to 1 is increased when the predicted output is less than the target output ($y < \hat{y}_i$) by providing Type I feedback. Type II feedback, on the other hand, is applied to decrease the number of clauses that evaluates to 1 when the predicted output is higher than the target output ($y > \hat{y}_i$).

Activation Probability: Feedback is handed out stochastically to regulate learning. The feedback probability p_j is proportional to the absolute error of the prediction, $\mid y - \hat{y}_i \mid$. Clauses activated for feedback are the stored in the matrix $\mathbf{P} = (p_j) \in \{0, 1\}^m$.

Type I Feedback: Type I feedback subdivides into Type Ia and Type Ib. Type Ia reinforces *include* actions of TAs whose corresponding literal value is 1, however, only when the clause output is 1. The probability of k^{th} TA of the j^{th} clause receives Type Ia feedback $r_{j,k}$ is $\frac{s-1}{s}$, where s (s ≥ 1) is a user set parameter. Type Ib combats over-fitting by reinforcing *exclude* actions of TAs when the corresponding literal is 0 or when the clause output is 0. The probability of k^{th} TA of the j^{th} clause receives Type Ib feedback $q_{j,k}$ is $\frac{1}{s}$.

Using the complete set of conditions, the TAs selected for Type Ia feedback are singled out by the indexes $I^{\mathrm{Ia}} = \{(j, k)|l_k = 1 \land c_j = 1 \land p_j = 1 \land r_{j,k} = 1\}$. Similarly, TAs selected for Type Ib are $I^{\mathrm{Ib}} = \{(j, k)| (l_k = 0 \lor c_j = 0) \land p_j = 1 \land q_{j,k} = 1\}$.

Once the TAs have been targeted for Type Ia and Type Ib feedback, their states are updated. Available updating operators are $\oplus$ and $\ominus$, where $\oplus$ adds 1 to the current state while $\ominus$ subtracts 1. Thus, before a new learning iterations starts, the states in the matrix $\mathbf{A}$ are updated as follows: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I^{\mathrm{Ia}}\right) \ominus I^{\mathrm{Ib}}$.

Type II Feedback: Type II feedback eventually changes the output of a clause from 1 to 0, for a specific input $\hat{X}_i$. This is achieved simply by including one or more of the literals that take the value 0 for $\hat{X}_i$. The indexes of TAs selected for Type II can thus be singled out as $I^{\mathrm{II}} = \{(j, k)|l_k = 0 \land c_j = 1 \land p_j = 1\}$. Accordingly, the states of the TAs are updated as follows: $\mathbf{A} \leftarrow \mathbf{A} \oplus I^{\mathrm{II}}$.

3 Stochastic Searching on the Line

Stochastic searching on the line, also referred to as stochastic point location (SPL) was pioneered by Oommen in 1997 [9]. SPL is a fundamental optimization problem where one tries to locate an unknown unique point within a given interval. The only available information for the Learning Mechanism (LM) is the possibly faulty feedback provided by the attached environment (E). According to the feedback, LM moves right or left from its current location in a discretized solution space.

The task at hand is to determine the optimal value λ^* of a variable λ, assuming that the environment is informative. That is, that it provides the correct direction of λ^* with probability $p > 0.5$. In SPL, λ is assume to be any number in the interval $[0, 1]$. The SPL scheme of Oommen discretizes the solution space by subdividing the unit interval into N steps, $\{0, 1/N, 2/N, ..., (N-1)/N, 1\}$. Hence, N defines the resolution of the learning scheme.

The current guess, $\lambda(n)$, is updated according to the feedback from the environment as follows:

$$\lambda(n+1) = \begin{cases} \lambda(n) + 1/N, & \text{if} \quad E(n) = 1 \text{ and } 0 \leqslant \lambda(n) < 1 \ , \\ \lambda(n) - 1/N, & \text{if} \quad E(n) = 0 \text{ and } 0 < \lambda(n) \leqslant 1 \ , \\ \lambda(n), & \text{Otherwise} \quad . \end{cases} \tag{4}$$

The feedback $E(n) = 1$ is the environment suggestion to increase the value of λ and $E(n) = 0$ is the environment suggestion to decrease the value of λ. Asymptotically, the learning mechanics is able to find a value arbitrarily close to λ^* when $N \to \infty$ and $n \to \infty$.

4 Regression Tsetlin Machine with Weighted Clauses

We now introduce clauses with integer weights to provide a more compact representation of the regression function. The regression function for the integer weighted RTM attaches a weight w_j to each clause output c_j, $j = 1, ..., m$. Consequently, the regression output can be computed according to Eq. 5:

$$y = \frac{1}{T} \sum_{j=1}^{m} w_j \prod_{k \in I_j} l_k . \tag{5}$$

Weight Learning: Our approach to learning the weight of each clause is similar to SPL. However, the solution space of each weight is $[0, \infty]$, while the resolution of the learning scheme is $N = 1$. The weight attached to a clause is updated when the clause receives Type Ia feedback or Type II feedback. The weight updating procedure is summarized in Algorithm 1. Here, $w_j(n)$ is the weight of clause j at the n^{th} training round.

```
Algorithm 1: Round $n$ updating of clause weights
Initialization (round 0): $w_j(0) \leftarrow 0, j = 1, \ldots, m$
Initialization (round $n$): $y$ is calculated according to Eq. 5.
for $j = 1, ..., m$ do
  if $y(n) < \hat{y}_i(n) \wedge c_j(n) = 1 \wedge p_j(n) = 1$ then
    $w_j(n+1) \leftarrow w_j(n) + N$
  else if $y(n) > \hat{y}_i(n) \wedge c_j(n) = 1 \wedge p_j(n) = 1 \wedge w_j(n) > 0$ then
    $w_j(n+1) \leftarrow w_j(n) - N$
  else
    $w_j(n+1) \leftarrow w_j(n)$
  end if
end for
Return $w_j(n+1), j = 1, \ldots, m$
```

Note that since weights in this study can take any value higher than or equal to 0, an unwanted clause can be turned off by setting its weight to 0. Further, sub-patterns that have a large impact on the calculation of y can be represented with a correspondingly larger weight.

5 Empirical Evaluation

In this section, we study the behavior of the RTM with integer weighting (RTM-IW) using two artificial datasets similar to the datasets presented in [1], in comparison with a standard RTM and a real-value weighted RTM (RTM-RW). We use Mean Absolute Error (MAE) to measure performance.

Artificial Datasets: Dataset I contains 3-bit feature input. The output, in turn, is 100 times larger than the decimal value of the binary input (e.g., the input [0, 1, 0] produces the output 200). The training set consists of 8000 samples while the 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. Each input feature has been generated independently with equal probability of taking either the value 0 or 1, producing a uniform distribution of bit values.

Results and Discussion: The pattern distribution of the artificial data was analyzed in the original RTM study. As discussed, there are eight unique sub-patterns. The RTM is able to capture the complete set of sub-patterns utilizing no more than three types of clauses, i.e., $(1 \ast \ast)$, $(\ast\ 1 \ast)$, $(\ast \ast\ 1)$[1]. However, to produce the correct output, some clauses must be duplicated multiple times, depending on the input pattern. For instance, each dataset requires *seven* clauses to represent the three different patterns it contains, namely, $(4 \times (1 \ast \ast), 2 \times (\ast\ 1 \ast), 1 \times (\ast \ast\ 1))$[2]. So, with e.g. the input [1, 0, 1], four clauses which represent the pattern $(1 \ast \ast)$ and one clause which represents the pattern $(\ast \ast\ 1)$ activate to correctly output 500 (after normalization).

Notably, it turns out that the RTM-IW requires even fewer clauses to capture the sub-patterns in the above data, as outlined in Table 1. Instead of having multiple clauses to represent one sub-pattern, RTM-IW utilizes merely one clause with the correct weight to do the same job. The advantage of the proposed integer weighting scheme is thus apparent. It learns the correct weight of each clause, so that it achieves an MAE of zero. Further, it is possible to ignore redundant clauses simply by giving them the weight zero. For the present dataset, for instance, decreasing m while keeping the same resolution, $T = 7$, does not impede accuracy. The RTM-RW on the other hand struggles to find the correct weights, and fails to minimize MAE. Here, the real valued weights were updated

[1] Here, $\ast$ means an input feature that can take an arbitrary value, either 0 or 1.

[2] In this expression, "*four* clauses to represent the pattern $(1 \ast \ast)$" is written as "4 $\times$ $(1 \ast \ast)$".

Table 1. Behavior comparison of different RTM schemes on Dataset III.

	m	T	Pattern	I_j	$\bar{I}_j$	No. of clauses required	w_j	Training MAE	Testing MAE
RTM	7	7	$(1 \ast \ast)$	{1}	{ }	4	–	0	0
			$(\ast\ 1 \ast)$	{2}	{ }	2	–		
			$(\ast \ast\ 1)$	{3}	{ }	1	–		
RTM-IW	3	7	$(1 \ast \ast)$	{1}	{ }	1	4	0	0
			$(\ast\ 1 \ast)$	{2}	{ }	1	2		
			$(\ast \ast\ 1)$	{3}	{ }	1	1		
RTM-RW	3	7	$(1 \ast \ast)$	{1}	{ }	1	3.987	1.857	1.799
			$(\ast\ 1 \ast)$	{2}	{ }	1	2.027		
			$(\ast \ast\ 1)$	{3}	{ }	1	0.971		

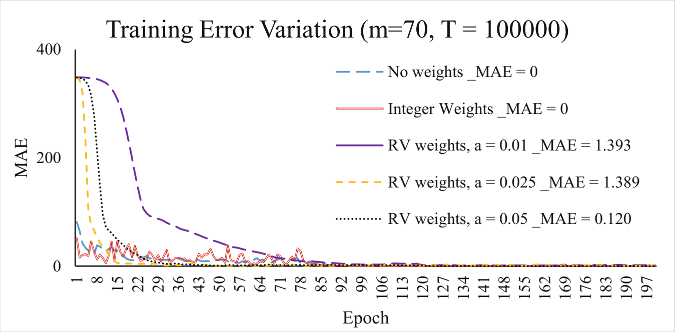

Fig. 1. The training error variation per training epoch for different RTM schemes.

with a learning rate of $\alpha = 0.01$, determined using a binary hyper-parameter search.

Figure 1 casts further light on learning behaviour by reporting training error per epoch for the three different RTM schemes with $m = 70$ and $T = 100000$. As seen, both RTM and RTM-IW obtain relatively low MAE after just one training epoch, eventually reaching MAE zero (training MAE at end of training are given in the legend of each graph). RTM-RW, on the other hand, starts off with a much higher MAE, which is drastically decreasing over a few epochs, however, fails to reach MAE 0 after becoming asymptotically stable.

We also studied the effect of T on performance with noise free data by varying T, while fixing the number of clauses m. For instance, RTM was able to reach a training MAE of 1.9 and a testing error of 2.1 with $m = T = 300$ [1]. For the same dataset, RTM-IW can reach a training error of 0.19 and a testing error of 1.87 with $m = 200$ and $T = 2000$. Further, for $m = 200$ and $T = 20\,000$, training error drops to 0.027 and testing error drops to 0.027. Finally, by increasing T to 200 000 training error falls to 0.0003 while testing error stabilises at 0.0002.

Table 2. Training and testing MAE after 200 training epochs by various methods with different m and T.

Model		RTM				RTM-RW				RTM-IW			
MAE		Training		Testing		Training		Testing		Training		Testing	
Dataset		I	II	I	II	I	II	I	II	I	II	I	II
m	7	0.000	7.400	0.000	5.000	2.230	7.702	2.217	5.955	1.172	8.019	1.171	6.236
	20	14.600	13.800	14.200	14.500	1.023	7.863	1.036	6.007	0.487	9.844	0.493	8.499
	70	0.000	6.600	0.000	4.200	0.292	7.365	0.295	5.735	0.189	7.602	0.189	5.532
	300	1.900	5.800	2.100	3.300	0.104	5.800	0.106	2.226	0.078	5.685	0.078	2.234
	700	1.000	5.900	1.000	3.400	0.013	5.551	0.013	1.968	0.044	5.532	0.044	2.149
	2000	1.000	5.600	1.200	1.900	0.012	5.731	0.012	2.520	0.003	5.373	0.003	1.280
	5000	0.900	5.500	1.000	2.700	0.010	5.635	0.010	2.252	0.001	5.412	0.001	1.501

To further compare the performance of RTM-IW with RTM and RTM-RW, each approach was evaluated using a wide rage of m and T settings. Representative training and testing MAE for both datasets are summarized in Table 2. Here, the number of clauses used with each dataset is also given. The T for the original RTM is equal to the number of clauses, while for the RTM with weights T is simply 100 times that number.

As seen, the training and testing MAE reach zero when the RTM operates with noise free data when $m = 7$. However, MAE approaches zero with RTM-IW and RTM-RW when increasing number of clauses m.

For noisy data, the minimum training MAE achieved by RTM is 5.500, obtained with $m = 5000$ clauses. The RTM-IW, on the other hand, obtains a lower MAE of 5.373 with less than half of the clauses ($m = 2000$). The accuracy of RTM-IW in comparison with RTM-RW is less clear, with quite similar MAE for noisy data. The average testing MAE across both the datasets, however, reveals that the average MAE of RTM-IW is lower than that of the RTM-RW (2.101 vs 2.168).

6 Conclusion

In this paper, we presented a new weighting scheme for the Regression Tsetlin Machine (RTM), RTM with Integer Weights (RTM-IW). The weights attached to the clauses helps the RTM represent sub-patterns in a more compact way. Since the weights are integer, interpretability is improved through a more compact representation of the clause set. We also presented a new weight learning scheme based on stochastic searching on the line, integrated with the Type I and Type II feedback of the RTM. The RTM-IW obtains on par or better accuracy with fewer number of clauses compared to RTM without weights. It also performs competitively in comparison with an alternative RTM with real-valued weights.

References

1. Abeyrathna, K.D., Granmo, O.-C., Jiao, L., Goodwin, M.: The Regression tsetlin machine: a tsetlin machine for continuous output problems. In: Moura Oliveira, P., Novais, P., Reis, L.P. (eds.) EPIA 2019. LNCS (LNAI), vol. 11805, pp. 268–280. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30244-3_23
2. Abeyrathna, K.D., Granmo, O.-C., Zhang, X., Jiao, L., Goodwin, M.: The regression Tsetlin machine: a novel approach to interpretable nonlinear regression. Philos. Trans. R. Soc. A **378**, 20190165 (2019)
3. Granmo, O.-C.: The tsetlin machine - a game theoretic bandit driven approach to optimal pattern recognition with propositional logic. arXiv:1804.01508
4. Phoulady, A., Granmo, O.-C., Gorji, S.R., Phoulady, H.A.: The weighted tsetlin machine: compressed representations with clause weighting. In: Ninth International Workshop on Statistical Relational AI (StarAI 2020) (2020)
5. Berge, G.T., Granmo, O.-C., Tveit, T.O., Goodwin, M., Jiao, L., Matheussen, B.V.: Using the Tsetlin Machine to learn human-interpretable rules for high-accuracy text categorization with medical applications. IEEE Access **7**, 115134–115146 (2019)
6. Wheeldon, A., Shafik, R., Yakovlev, A., Edwards, J., Haddadi, I., Granmo, O.-C.: Tsetlin machine: a new paradigm for pervasive AI. In: Proceedings of the SCONA Workshop at Design, Automation and Test in Europe (DATE) (2020)
7. Granmo, O.-C., Glimsdal, S., Jiao, L., Goodwin, M., Omlin, C. W., Berge, G.T.: The convolutional tsetlin machine. arXiv preprint:1905.09688 (2019)
8. Rahimi Gorji, S., Granmo, O.-C., Phoulady, A., Goodwin, M.: A tsetlin machine with multigranular clauses. In: Bramer, M., Petridis, M. (eds.) SGAI 2019. LNCS (LNAI), vol. 11927, pp. 146–151. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34885-4_11
9. Oommen, B.J.: Stochastic searching on the line and its applications to parameter learning in nonlinear optimization. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **27**(4), 733–739 (1997)
10. Tsetlin, M.L.: On behaviour of finite automata in random medium. Avtomat. i Telemekh **22**(10), 1345–1354 (1961)