On the Convergence of Tsetlin Machines for the XOR Operator

Lei Jiao, *Senior Member, IEEE*, Xuan Zhang, Ole-Christoffer Granmo, and K. Darshana Abeyrathna

Abstract—The Tsetlin Machine (TM) is a novel machine learning algorithm with several distinct properties, including transparent inference and learning using hardware-near building blocks. Although numerous papers explore the TM empirically, many of its properties have not yet been analyzed mathematically. In this article, we analyze the convergence of the TM when input is non-linearly related to output by the XOR-operator. Our analysis reveals that the TM, with just two conjunctive clauses, can converge almost surely to reproducing XOR, learning from training data over an infinite time horizon. Furthermore, the analysis shows how the hyper-parameter T guides clause construction so that the clauses capture the distinct sub-patterns in the data. Our analysis of convergence for XOR thus lays the foundation for analyzing other more complex logical expressions. These analyses altogether, from a mathematical perspective, provide new insights on why TMs have obtained the state-of-the-art performance on several pattern recognition problems.

Index Terms—Tsetlin Automata, Propositional Logic, Tsetlin Machine, Convergence Analysis, XOR Operator

1 Introduction

The emerging paradigm of Tsetlin machines (TM) [1], initiated in 2018, makes a shift from arithmetic-based to logic-based machine learning [2]. Like logical engineering, a TM produces propositional/relational Horn clauses [3]. However, the logical expressions are robustly learnt using finite state machines in the form of Tsetlin automata (TAs) [4]. Via a game-theoretic collaboration scheme, the TAs self-organize to capture the distinct patterns in the data. The dynamics of the collaboration involves three interacting mechanisms. High pattern recall is enforced by a resource allocation mechanism that diversifies clause construction. Simultaneously, a mechanism that forces the clauses to capture frequent patterns combats overfitting. Finally, without compromising high pattern frequency, the discrimination power of the clauses is optimized by injecting discriminative features.

TMs provide two main advantages: transparent inference and learning combined with hardware-near building blocks. TM transparency, which unravels the reasoning behind the decision making process, addresses one of the most critical challenges in Artificial Intelligence (AI) research – lack of interpretability [5]. In particular, deep learning-based approaches mainly employ post-processing for *approximate* local interpretation of individual predictions, which do not guarantee model fidelity [6]. TMs, on the other hand, is founded on conjunctive clauses in propositional logic, which have been postulated as particularly easy for humans to comprehend [7]. TMs further facilitate derivation of closed formula expressions for both local and global interpretability, akin to SHAP [8]. Computationally, TMs can be realized via a set of finite-state automata — the TAs — which are well-suited for implementation in hardware, such as on FPGA [9]. Different from the extensive arithmetic operations required by most other AI approaches, a TA learns using increment and decrement operations only [4]. Indeed, due to the robustness of TA learning and TM pattern representation, the TM paradigm is shown to be inherently fault-tolerant, completely masking stuck-at faults [10].

There are many variations of TMs, with two main architectures being the convolutional TM (CTM) [11] and the regression TM (RTM) [12], [13]. The CTM has provided competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST, in comparison with K-Nearest Neighbor, Support Vector Machines, Random Forests, Gradient Boosting, BinaryConnect, Logistic Circuits, Convolutional Neural Networks and ResNet [11]. Similarly, RTM has compared favorably with Regression Trees, Random Forest Regression, and Support Vector Regression [13]. The above TM approaches have further been enhanced by various modifications. By introducing real-valued clause weights, the number of clauses can be reduced by up to $50\times$ without accuracy loss [14]. Also, the logical inference structure of TMs makes it possible to index the clauses on the features that falsify them, increasing inference and learning speed by up to an order of magnitude [15]. In [16], stochastic searching on the line automata [17] learns integer clause weights, performing on-par or better than Random Forest, Gradient Boosting, Neural Additive Models, StructureBoost and Explainable Boosting Machines. In [18], a novel variant of the TM that randomly drops clauses has been proposed, from which we observe up to +10% increase in accuracy and $2\times$ to $4\times$ faster learning compared with vanilla TM. For natural language processing, the TM has also achieved competitive results in text classification [19], [20], word sense disambiguation [21], novelty detection [22], [23], fake news

- *Lei Jiao, Ole-Christoffer Granmo, and K. Darshana Abeyrathna are with the Centre for Artificial Intelligence Research, University of Agder, 4879, Grimstad, Norway.*
  E-mail: lei.jiao@uia.no
- *Xuan Zhang is with the Norwegian Research Centre (NORCE), 4879, Grimstad, Norway.*
  E-mail: xuzh@norceresearch.no

Manuscript received XXX; revised XXX.

detection [24], semantic relation analysis [25], aspect-based sentiment analysis [26], and robustness towards counterfactual data [27]. Lately, more advanced architectures have appeared, such as the relational TM [3] and the coalesced TM [28].

The above studies report that TMs, with smaller memory footprint and higher computational efficiency, obtain better or competitive classification and regression accuracy compared with most of the state-of-the-art AI techniques, while maintaining transparency. Although numerous papers explore the TM empirically, many of its properties have not yet been analyzed mathematically. In [29], convergence for unary operators on one-bit data, i.e., the IDENTITY- and the NOT operators, is analyzed. There, we first proved that the TM can converge almost surely to the intended pattern when the training data is noise-free. Thereafter, we analyzed the effect of noise, establishing how the noise probability of the data and the granularity hyper-parameter, i.e., s, of the TM govern convergence [29].

Paper Contributions. In this paper, we analyze the "XOR" case, which deals with the binary XOR operator, encompassing two critical sub-patterns. We start from a simple structure of two clauses, each of which has four TAs with only two states. For this structure, we prove convergence via discrete time Markov chain (DTMC) analysis, analysing the ability of TMs to learn the XOR operator from data. Thereafter, we investigate the convergence behavior for more than two clauses. From the latter analysis, we reveal the crucial role the hyper-parameter T of TMs plays, showing how this parameter controls the ability to robustly capture multiple sub-patterns within one class, through allocating sparse pattern representation resources (the clauses).

Paper Organization. The remainder of the paper is organized as follows. Section 2 briefly reviews the TM and specifies the training process for XOR. In Section 3, we present our analytical procedure and the main analytical results. We conclude the paper in Section 5.

2 Review of the Tsetlin Machine

In this section, we present the TM in brief, including an overview of TA, the TM architecture, and the training process of TMs. A more comprehensive exposition can be found in [1].

2.1 Tsetlin Automata (TA)

A TA is a fixed structure deterministic learning automaton [30], [31], forming a crucial component of TM learning. By interacting with the environment, a TA aims to learn the action that offers the highest probability of providing a reward [4]. Figure 1 illustrates a two-action TA with $2N$ states, where $N \in [1, +\infty)$. Which action a TA selects is decided by its current state, which triggers a response from the environment followed by the TM making a state transition. That is, when the TA is in states 0 to $N-1$, i.e., on the left-hand side of the state-space shown in Figure 1, Action 1 is chosen. If the TA on the other hand finds itself in states N to $2N-1$, i.e., on the right-hand side, Action 2 is chosen. Once an action is chosen, the environment responds with either a reward or a penalty. When the TA receives a penalty, it will move towards the opposite half of the state space, that is, towards the other action. This transition is marked by the solid arrows in Figure 1. Conversely, if the TA receives a reward, it will switch to a "deeper" state by transitioning to the left or the right end of the chain, depending on whether the current action is Action 1 or Action 2. In the figure, this transition is captured by the dashed arrows. Note that the number of states in a TA, i.e., $2N$, can be adjusted. The larger the number, the slower the convergence. However, the TA learns more accurately in a stochastic environment with a larger number of states.

2.2 Tsetlin Machines (TMs)

A TM is formed by m teams of TAs. The TAs operate on binary input and employ propositional logic to represent patterns. In general, the input of a TM can be represented by $\mathbf{X} = [x_1, x_2, \ldots, x_o]$, with $x_k \in \{0,1\}, k = 1, 2, \ldots, o$. Each TA team contains o pairs of TAs, with each pair being responsible for a certain input variable x_k. Figure 2 shows such a TA team $\mathcal{G}_j^i = \{\mathrm{TA}_{k'}^{i,j} | 1 \leq k' \leq 2o\}$ that has $2o$ TAs. The index i refers to a specific pattern class and j is the index of a specific clause. The automaton $\mathrm{TA}_{2k-1}^{i,j}$ returns the input x_k as is, whereas $\mathrm{TA}_{2k}^{i,j}$ addresses the negation of x_k, i.e., $\neg x_k$. Note that the inputs and their negations are jointly referred to as literals.

Each TA chooses one of two actions, i.e., it either "Includes" or "Excludes" its literal, outputting $I(\cdot)$ and $E(\cdot)$, respectively. Let $I(x) = x$, $I(\neg x) = \neg x$, and $E(\cdot) = 1$, with the latter meaning that an excluded literal does not contribute to the output. Collectively, the $I(\cdot)/E(\cdot)$-outputs of the TA team then take part in a conjunction, expressed by the conjunctive clause [29]:

$$C_j^i(\mathbf{X}) = \begin{cases} \left(\bigwedge\limits_{k \in I_j^i} x_k\right) \wedge \left(\bigwedge\limits_{k \in \bar{I}_j^i} \neg x_k\right) \wedge 1 & \text{For training,} \\ \left(\left(\bigwedge\limits_{k \in I_j^i} x_k\right) \wedge \left(\bigwedge\limits_{k \in \bar{I}_j^i} \neg x_k\right)\right) \vee 0 & \text{For testing.} \end{cases} \tag{1}$$

In Eq. (1), I_j^i and $\bar{I}_j^i$ are the subsets of indexes for the literals that have been included in the clause. I_j^i contains the indexes of included non-negated inputs, x_k, whereas $\bar{I}_j^i$ contains the indexes of included negated inputs, $\neg x_k$. The "0" and "1" in Eq. (1) make sure that $C_j^i(\mathbf{X})$ also is defined when all the TAs choose to exclude their literals. As can be observed, during training, an "empty" clause outputs 1, while it outputs 0 during testing (operation).

Multiple TA teams, i.e., clauses, are finally assembled into a complete TM. There are two architectures for clause assembling: Disjunctive Normal Form Architecture and Voting Architecture. In this study, we focus on the latter one, as shown in Figure 3. For this architecture, the voting consists of summing the output of the clauses:

$$f_{\sum}(\mathcal{C}^i(\mathbf{X})) = \sum_{j=1}^{m} C_j^i(\mathbf{X}). \tag{2}$$

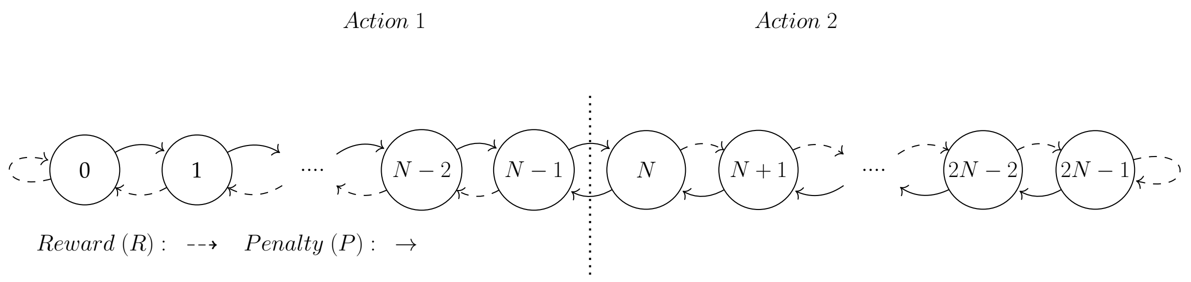


Fig. 1: A two-action Tsetlin Automaton with $2N$ states.

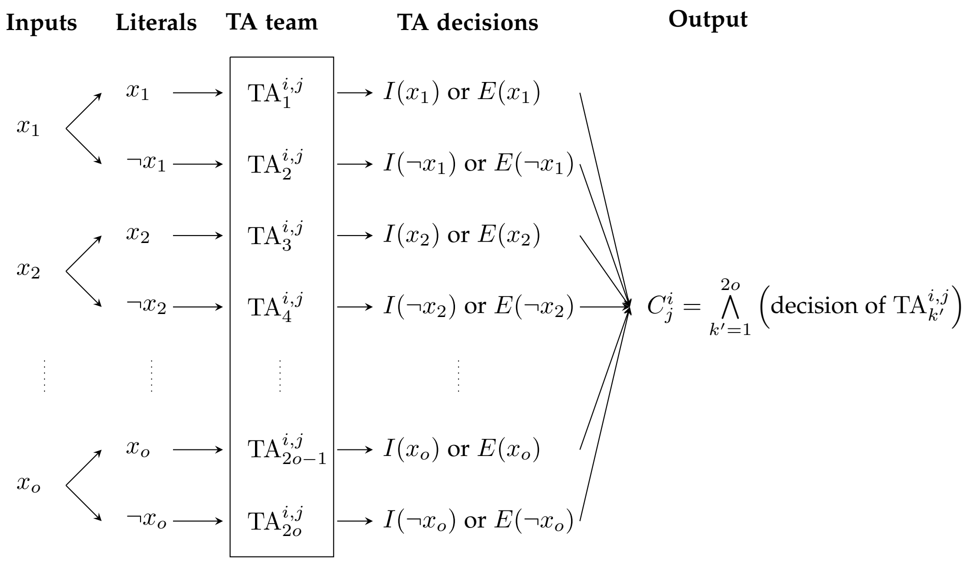


Fig. 2: A TA team G_j^i consisting of $2o$ TAs [29]. Here $I(x_1)$ means "include x_1" and $E(x_1)$ means "exclude x_1".

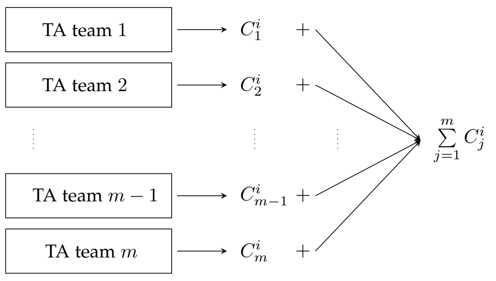


Fig. 3: TM voting architecture.

The output of the TM, in turn, is decided by the unit step function:

$$\hat{y}^i = \begin{cases} 0 & \text{for } f_{\sum}(\mathcal{C}^i(\mathbf{X})) < Th \\ 1 & \text{for } f_{\sum}(\mathcal{C}^i(\mathbf{X})) \geq Th \end{cases}, \quad (3)$$

where Th is a predefined threshold for classification. Note that for this architecture, the TM can assign a polarity to each TA team [1]. For example, TA teams with odd indexes get positive polarity, and they vote for class i. The remaining TA teams get negative polarity and vote against class i. The voting consists of summing the output of the clauses, according to polarity, and the threshold Th is configured as zero. In this study, for ease of analysis, we consider only positive polarity clauses. Nevertheless, this does not change the nature of TM learning (negative polarity clauses simply "invert" the feedback given to them).

2.3 The Tsetlin Machine Game for Learning Patterns

2.3.1 *The Tsetlin Machine Game*

The TM trains the TA teams, associated with the clauses, to make the clauses C_j^i, $j = 1, 2, ..., m$, capture the sub-patterns that characterize the class i. Data $(\mathbf{X} = [x_1, x_2, ..., x_o],\ y^i)$ for training is obtained from a dataset $\mathcal{S}$, distributed according to the probability distribution $P(\mathbf{X}, y^i)$. The training process is built on letting all the TAs take part in a decentralized game. In the game, each TA is guided by Type I Feedback and Type II Feedback defined in Table 1 and Table 2, respectively. Type I Feedback is triggered when the training sample has a positive label, i.e., $y^i = 1$, meaning that the sample belongs to class i. When the training sample is labeled as not belonging to class i, i.e., $y^i = 0$, Type II Feedback is utilized for generating responses. These two types of feedback are designed to reinforce true positive output, i.e., $(\hat{y}^i = 1, y^i = 1)$ and true negative output, i.e., $(\hat{y}^i = 0, y^i = 0)$. Simultaneously, they suppress false positive, i.e., $(\hat{y}^i = 1, y^i = 0)$, and false negative output, i.e., $(\hat{y}^i = 0, y^i = 1)$.

The formation of patterns is founded on frequent pattern mining. That is, a parameter s controls the granularity of the clauses. A larger s allows more literals to be included in each clause, making the corresponding sub-patterns more fine-grained. A more detailed analysis on parameter s can be found in [29].

Value of the clause $C_j^i(\mathbf{X})$		1		0	
Value of the Literal $x_k/\neg x_k$		1	0	1	0
TA: **Include Literal**	P(Reward)	$\frac{s-1}{s}$	NA	0	0
	P(Inaction)	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
TA: **Exclude Literal**	P(Reward)	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	P(Inaction)	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	$\frac{s-1}{s}$	0	0	0

TABLE 1: Type I Feedback — Feedback upon receiving a sample with label $y = 1$, for a single TA to decide whether to Include or Exclude a given literal $x_k/\neg x_k$ into C_j^i. NA means not applicable [11].

Value of the clause $C_j^i(\mathbf{X})$		1		0	
Value of the Literal $x_k/\neg x_k$		1	0	1	0
TA: **Include Literal**	P(Reward)	0	NA	0	0
	P(Inaction)	1.0	NA	1.0	1.0
	P(Penalty)	0	NA	0	0
TA: **Exclude Literal**	P(Reward)	0	0	0	0
	P(Inaction)	1.0	0	1.0	1.0
	P(Penalty)	0	1.0	0	0

TABLE 2: Type II Feedback — Feedback upon receiving a sample with label $y = 0$, for a single TA to decide whether to Include or Exclude a given literal $x_k/\neg x_k$ into C_j^i. NA means not applicable [11].

To avoid the situation that a majority of the TA teams single in on only a subset of the patterns in the training data, forming an incomplete representation, we use a hyper-parameter T as target for the summation $f_{\sum}$. If the votes for a certain sub-pattern accumulate to a total of T or more, neither rewards or penalties are provided to the TAs when more training samples of this sub-pattern are given. In this way, we can ensure that only a few of the available clauses are utilized to capture each specific sub-pattern. In more details, the strategy works in the manner below:

Generating Type I Feedback. If the output of the training sample is $y^i = 1$, we generate *Type I Feedback* for each clause $C_j^i \in \mathcal{C}^i$, where $\mathcal{C}^i$ is the set of clauses that are trained for pattern i, however, not every time. Instead, the decision to give feedback to a specific clause is random, according to a feedback probability. The probability of generating Type I Feedback is [1]:

$$u_1 = \frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}_i)))}{2T}. \tag{4}$$

Generating Type II Feedback. If the output of the training sample is $y^i = 0$, we generate *Type II Feedback* to each clause $C_j^i \in \mathcal{C}^i$, again randomly. The probability of generating Type II Feedback is [1]:

$$u_2 = \frac{T + \max(-T, \min(T, f_{\sum}(\mathcal{C}_i)))}{2T}. \tag{5}$$

After Type I Feedback or Type II Feedback have been triggered for a clause, the individual TA within each clause is given reward/penalty/inaction according to the probability defined, and then the system is updated.

x_1	x_2	y
0	0	0
1	1	0
0	1	1
1	0	1

TABLE 3: The "XOR" logic.

2.3.2 The Training Process in the XOR Case

We now introduce the special case of training TMs to capture XOR-patterns. We assume that the training samples shown in Table 3 are provided without noise. In other words, we have $P(y = 1|x_1 = 0, x_2 = 1) = 1$, $P(y = 1|x_1 = 1, x_2 = 0) = 1$, $P(y = 0|x_1 = 0, x_2 = 0) = 1$, and $P(y = 0|x_1 = 1, x_2 = 1) = 1$. We also assume that $P(x_1 = 0, x_2 = 1) > 0$, $P(x_1 = 1, x_2 = 1) > 0$, $P(x_1 = 0, x_2 = 0) > 0$, and $P(x_1 = 1, x_2 = 0) > 0$. Clearly $P(x_1 = 0, x_2 = 1)+P(x_1 = 1, x_2 = 1)+P(x_1 = 0, x_2 = 0)+P(x_1 = 1, x_2 = 0) = 1$. This guarantees that all types of possible input-output pairs will appear in the training samples. The aim is to show that after training, the TM can output 1 for inputs $x_1 = 1, x_2 = 0$ or $x_1 = 0, x_2 = 1$, and 0 otherwise.

The above XOR-scenario leads to the following TM training process, described step-by-step:

1) We initialize the TAs by assigning each of them a random state among the states associated with action Exclude.
2) We obtain a new training sample (x_1, x_2, y) and calculate the value of each single clause C_j^i according to Eq. (1).
3) The TA states for each clause are updated based on: (i) the label y; (ii) the clause value C_j^i; (iii) the value of each individual literal (x_1, $\neg x_1$, x_2, $\neg x_2$,); and (iv) the sum of the clause outputs for the class i, $f_{\sum}(\mathcal{C}^i(\mathbf{X}))$. Finally, for each clause, the the states of the associated TAs are updated according to Table 1 with probability u_1 when $y = 1$. If $y = 0$, the TAs are updated according to Table 2, with probability u_2.
4) Repeat from Step 2 until a given stopping criteria is met.

Note that in the XOR case, there is only one class to be learnt, which is the XOR-relation. We therefore ignore the class index, i.e., i, in notation C_j^i and $\text{TA}_{2k}^{i,j}$ in the remainder of the paper.

For XOR-relation, as can be seen from Table 3, the inputs for the two to-be-learnt sub-patterns ($x_1 = 1, x_2 = 0$ and $x_1 = 0, x_2 = 1$) are the bit-wise inversion of the other. Therefore, the XOR relation is not linearly separable (in 2D) and thus a "single-layer" perceptron cannot implement XOR [32]. In the training process of TM, different clauses need to be guided to learn opposite propositional expressions in order to follow both sub-patterns required by XOR. The task seems to be difficult as the training samples that reinforce the convergence of one sub-pattern will discourage the convergence of the other sub-pattern. This motivates us to study the convergence behavior of XOR operator in more details in order to understand how TMs learn opposite sub-patterns. In fact, it is the hyper-parameter T that plays the significant role via Eqs. (4) and (5), which is to be revealed through the proof.

3 Proof of the Convergence for the XOR Operator

In what follows, we will reveal, step-by-step, the convergence property of the TM for the XOR-relation. First, in Subsection 3.1, we start from a special and simple case to show that there exists a TM configuration that can learn the XOR-relation. This establishes that TMs have the ability to learn such a relation. Thereafter, in Subsection 3.2, we analyze how a general TM can learn the XOR-relation, including the criteria for learning. Through these analyses, the dynamics of the learning process of the TAs, operating within the TM clauses, are elaborated. In particular, we investigate the self-organizing collaboration that happens among the clauses, to cast light on how a TM learns multiple sub-patterns.

3.1 The Simplest Structure for the XOR-relation

Theorem 1. *There exists a TM structure that can converge almost surely to the XOR-relation under an infinite time horizon.*

Proof. To prove Theorem 1, we use a TM with two clauses, C_1 and C_2. In C_1, there are four literals, i.e., x_1, $\neg x_1$, x_2, and $\neg x_2$, each of which corresponds to a TA, namely, TA^1_1, TA^1_2, TA^1_3, and TA^1_4. Similarly, in C_2, there are also four literals, i.e., x_1, $\neg x_1$, x_2, and $\neg x_2$, each of which corresponds to four other TAs, namely, TA^2_1, TA^2_2, TA^2_3, and TA^2_4. Clearly, there are in total 8 TAs in the system. Considering the simplest structure for TA, we provide each TA with only two states, as shown in Figure 4.

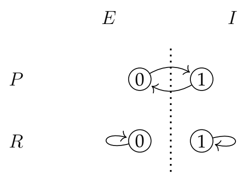


Fig. 4: A simple TA with two states. In this figure, "P", "R", "I", and "E" means "penalty", "reward", "include" and "exclude" respectively.

The behavior of the above depicted TM can be modeled using a discrete time Markov chain (DTMC) with 8 elements, each of which represents the status of the corresponding TA. In more details, any state of the DTMC is represented by $\boldsymbol{x} = (h_1, h_2, h_3, \ldots, h_8)$, where $h_i \in \{0, 1\}$, $i \in \{1, \ldots, 8\}$. Here, $h_1, h_2, h_3, \ldots, h_8$ correspond to TA^1_1, TA^1_2, TA^1_3, TA^1_4 TA^2_1, TA^2_2, TA^2_3, and TA^2_4. For example, h_1 represents the state for TA^1_1, with state 0 referring to "Exclude" and state 1 referring to "Include". This is also how the other h_i are organized. The state space of the DTMC, $\mathcal{S}$, includes $2^8 = 256$ states. If the system can capture the XOR-relation after training, the DTMC must have and only have two possible absorbing states, i.e., $(1, 0, 0, 1, 0, 1, 1, 0)$ for $C_1 = x_1 \land \neg x_2$ and $C_2 = \neg x_1 \land x_2$, and $(0, 1, 1, 0, 1, 0, 0, 1)$ for $C_1 = \neg x_1 \land x_2$ and $C_2 = x_1 \land \neg x_2$. Let us index the states from $(0, 0, 0, 0, 0, 0, 0, 0)$ to $(1, 1, 1, 1, 1, 1, 1, 1)$ as 1 to 256. Then the states $(0, 1, 1, 0, 1, 0, 0, 1)$ and $(1, 0, 0, 1, 0, 1, 1, 0)$ correspond to the 106^{th} and the 151^{st} state, respectively.

To determine whether the two states are absorbing, we can observe the transition matrix of the DTMC and see if there are any out going transitions from those two states. This can be easily checked and confirmed. To demonstrate that these two states are the only absorbing states, we also need to show that all the other states are recurrent. To demonstrate this point in a simple way, we calculate the limiting matrix of the DTMC. In more details, we first compose the transition matrix of the DTMC, $\boldsymbol{P}$, and then find the limiting matrix $\boldsymbol{A} = \boldsymbol{P}^{\infty}$. If the matrix $\boldsymbol{A}$ possesses the below properties, we can conclude that state 106 and state 151 are the only absorbing states. This, in turn, means that after infinite training samples, the system will learn the XOR-relation with probability 1. The properties are as follows:

- The transition probability from the 106^{th} state to the 106^{th} state is 1, and the same applies to the 151^{st} state.
- The transition probabilities from any state other than the two absorbing ones to the two absorbing ones sum to 1.
- The transition probabilities from any state other than the two absorbing ones to a non-absorbing state are all zeros.

The matrix $\boldsymbol{A}$ represents the probability of arriving at a destination state from any starting state after infinite time steps. The first bullet point shows that the 106^{th} and 151^{st} elements are indeed the absorbing states. This is because each of these states returns to itself with probability 1. Similarly, the second and the third bullet points indicate that the other states are not absorbing states because starting from any other state, the system will end up in one of the absorbing states.

In principle, $\boldsymbol{P}$ must be multiplied with itself an infinite number of times. In practice, however, we multiply $\boldsymbol{P}$ with itself a sufficiently large number of times, until the entries in $\boldsymbol{P}$ do not change.

To validate the convergence, we use the hyper-parameters $s = 10$ and $T = 1$ as an example[1] and use Algorithm 1 in Appendix 1 for the calculation[2]. In this example, we assume the training samples (1,1,0), (1,0,1) (0,1,1) and (1,1,0) appear with the same probability, i.e., 25% of the time each. From running the algorithm, we conclude that the 106^{th} and the 151^{st} are indeed the only absorbing states of the DTMC, which confirms that even the simplest configuration of the TM can converge almost surely to the XOR-relation. □

3.2 Structures with More Than Two TA States and/or More Than Two Clauses

Theorem 1 confirms that TMs are capable of learning the XOR-relation. In the following, we study the cases where there are more than two clauses and/or more than two TA

1. Here $T = 1$ is critical for the convergence, which is to be explained in the proof of Theorem 2.

2. The Python code for Algorithm 1 can be obtained from https://github.com/cair/TM-XOR-proof.

x_1	x_2	Output
0	0	0
1	1	0
0	1	1

TABLE 4: A sub-pattern in "XOR" case.

x_1	x_2	Output
0	0	0
1	1	0
1	0	1

TABLE 5: A sub-pattern in "XOR" case.

states. The purpose is to uncover how the XOR-relation is learnt by the TM in general. This also allows us to demonstrate the role that the hyper-parameter T plays during learning. We look in particular at how T governs reinforcement of the different clauses to learn the distinct sub-patterns associated with the XOR-relation.

The flow of the analysis is given via the lemmas and theorem below:

Lemma 1. *Any clause will converge almost surely to $\neg x_1 \land x_2$ given the training samples indicated in Table 4 in infinite time when $u_1 > 0$ and $u_2 > 0$.*

Lemma 2. *Any clause will converge almost surely to $x_1 \land \neg x_2$ given the training samples indicated in Table 5 in infinite time when $u_1 > 0$ and $u_2 > 0$.*

Lemma 3. *The system for any clause is recurrent given the input and output pair indicated in Table 3 for $u_1 > 0$ and $u_2 > 0$.*

Lemma 4. *Given a number of clauses m and a threshold value T, $T < m$, the event that the sum of the clause outputs, i.e., $f_{\sum}(\mathcal{C}_i)$, reaches T appears almost surely in infinite time.*

Lemma 5. *When the number of clauses that follow the same sub-pattern reaches T, other clauses will not see the training samples of this particular sub-pattern.*

Theorem 2. *The clauses can almost surely learn the sub-patterns of XOR in infinite time, when $T \leq m/2$.*

The reasoning and the connections of the above lemmas and theorem are outlined as follows. Lemma 1 confirms the fact that the TM can learn the intended sub-pattern if only one of the XOR sub-patterns, i.e., $x_1 = 0$, $x_2 = 1$ and $y = 1$, appears in the training data. Similarly, Lemma 2 confirms the convergence of TM for the other sub-pattern. In Lemmas 1 and 2, we assume non-negative u_1 and u_2 to guarantee that the training samples always trigger the feedback shown in Table 1 or Table 2. Note that these lemmas determine that the correct states are absorbing states as well as the uniqueness of these states. Lemma 3 establishes the fact that when both sub-patterns are presented in the training samples, a TM will not converge to any one of these in probability 1, if both u_1 and u_2 are kept positive. In other words, based on Lemmas 1-3, although TM will converge to the intended sub-pattern if only one sub-pattern appears in the training data, the system will not converge to any one of the sub-patterns if both sub-patterns appear during training. Therefore, it is necessary to have an additional mechanism to guide the convergence when both sub-patterns appear, which in fact is done by the hyper-parameter T via Eqs. (4) and (5). Indeed, the probability of triggering the feedback events in Table 1 or Table 2 is reduced while a sub-pattern becomes learnt, and eventually the corresponding training samples will be blocked to TM for any learnt sub-pattern. Specifically, Lemma 4 establishes that the number of clauses that learn a certain sub-pattern will reach T at a certain time instant. Lemma 5 guarantees that when the event described by Lemma 4 happens, the corresponding training samples of the learnt sub-pattern will be blocked from the system. In this way, only the training samples of the unlearnt sub-pattern will appear to TM so that the unlearnt sub-pattern can be learnt according to Lemma 1 or 2. To summarize, Lemmas 1-3 cover the system dynamics when T is not involved in the learning (and thus u_1 and u_2 are kept positive), while Lemmas 4 and 5 shows how T blocks the training samples of a learnt sub-pattern to make the system learn another sub-pattern. Based on Lemmas 1-5, we prove that the TM can learn the XOR-relation and the conditions for learning, in terms of Theorem 2. In the remaining subsections, we will prove them one by one.

3.2.1 Proof of Lemma 1 and Lemma 2

Now let's study Lemma 1. Here, we will confirm that the clauses in the TM will almost surely converge to the clause $\neg x_1 \land x_2$ when the training samples shown in Table 4 are given to the TM. Note that the functionality of T is disabled in this lemma, and u_1 and u_2 are assumed to be positive constants.

Proof. Without loss of generality, we study clause C_3, which has TA_1^3 with actions "Include" x_1 or "Exclude" it, TA_2^3 with actions "Include" $\neg x_1$ or "Exclude" it, TA_3^3 with actions "Include" x_2 or "Exclude" it, and TA_4^3 with actions "Include" $\neg x_2$ or "Exclude" it. To analyze the convergence of those four TAs, we perform a quasi-stationary analysis, where we freeze the behavior of three of them, and then study the transitions of the remaining one. More specifically, the analysis is organized as follows:

1) We freeze TA_1^3 and TA_2^3 respectively at "Exclude" and "Include". In this case, the first bit becomes $\neg x_1$. There are four sub-cases for TA_3^3 and TA_4^3:
    a) We study the transition of TA_3^3 when it has the action "Include" as its current action, given different training samples shown in Table 4 and different actions of TA_4^3 (i.e., when the action of TA_4^3 is frozen at "Include" or "Exclude").
    b) We study the transition of TA_3^3 when it has "Exclude" as its current action, given different training samples shown in Table 4 and different actions of TA_4^3 (i.e., when the action of TA_4^3 is frozen at "Include" or "Exclude").
    c) We study the transition of TA_4^3 when it has "Include" as its current action, given different training samples shown in Table 4 and different actions of TA_3^3 (i.e., when the action of TA_3^3 is frozen at "Include" or "Exclude").
    d) We study the transition of TA_4^3 when it has "Exclude" as its current action, given different training samples shown in Table 4 and

different actions of TA_3^3 (i.e., when the action of TA_3^3 is frozen as "Include" or "Exclude").

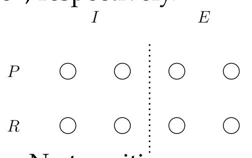
2) We freeze TA_1^3 and TA_2^3 respectively at "Include" and "Exclude". In this case, the first bit becomes x_1. The sub-cases for TA_3^3 and TA_4^3 are identical to the sub-cases in the previous case.
3) We freeze TA_1^3 and TA_2^3 at "Exclude" and "Exclude". In this case, the first bit is excluded and will not influence the final output. The sub-cases for TA_3^3 and TA_4^3 are identical to the sub-cases in the previous case.
4) We freeze TA_1^3 and TA_2^3 at "Include" and "Include". In this case, we always have $C_3 = 0$ because the clause contains the contradiction $x_1 \wedge \neg x_1$. The sub-cases for TA_3^3 and TA_4^3 are identical to the sub-cases in the previous case.

In the analysis below, we will study each of the four cases, one by one.

Case 1

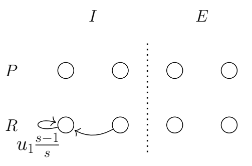
We now analyze the first sub-case, i.e., Sub-case 1 (a). In this case, the first bit is in the form of $\neg x_1$ always. We here study the transition of TA_3^3 when its current action is "Include". Depending on different training samples and actions of TA_4^3, we have the following possible transitions. Below, "I" and "E" mean "Include" and "Exclude", respectively.

Condition: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA_4^3=E.
Therefore, we have Type II feedback for literal $x_2 = 1$, clause $C_3 = 0$.

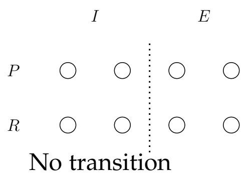


No transition

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA_4^3=E.
Therefore, we have Type I feedback for literal $x_2 = 1$, $C_3 = \neg x_1 \wedge x_2 = 1$.

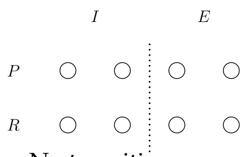


Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA_4^3=E.
Therefore, we have Type II feedback for literal $x_2 = 0$, $C_3 = \neg x_1 \wedge x_2 = 0$.



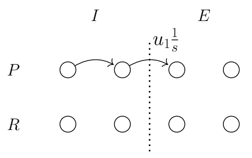
No transition

Condition: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA_4^3=I.
Therefore, we have Type II feedback for literal $x_2 = 1$, $C_3 = 0$.

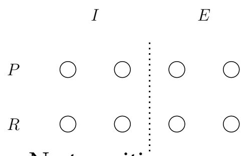


No transition

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA_4^3=I.
Therefore, we have Type I feedback for literal $x_2 = 1$, $C_3 = 0$.



Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA_4^3=I.
Therefore, we have Type II feedback for literal $x_2 = 0$, $C_3 = 0$.



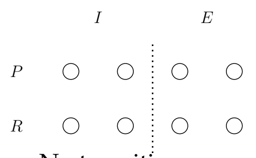
No transition

The above analyzed sub-case has 6 instances, depending on the variations of the training samples and the status of TA_4^3, where the first three correspond to the instances where TA_4^3=E while the last three represent the instances where TA_4^3=I. We now investigate the first instance, which covers the training samples: $x_1 = 1$, $x_2 = 1$, $y = 0$, and TA_4^3=E. This training sample will trigger Type II feedback because of $y = 0$. Then the clause becomes $C_3 = \neg x_1 \wedge x_2 = 0$ because the studied instance has TA_1^3=E, TA_2^3=I, TA_3^3=I, and TA_4^3=E. Because we now study TA_3^3, the corresponding literal is $x_2 = 1$. Based on the above information, we can check from Table 2 that the probability of "Inaction" is 1. Therefore, the transition diagram does not have any arrow, indicating that there is "No transition" for TA_3^3 in this circumstance.

To study a circumstance where transitions may happen, let us look at the second instance in the analyzed sub-case, with $x_1 = 0$, $x_2 = 1$, $y = 1$, and TA_4^3=E. This training sample will trigger Type I feedback as $y = 1$. Together with the current status of other TAs, the clause is determined to be $C_3 = \neg x_1 \wedge x_2 = 1$ and the literal is $x_2 = 1$. From Table 1, we know that the reward probability is $\frac{s-1}{s}$ and the inaction probability is $1/s$. To indicate the transitions, we have plotted the diagram showing the reward probability. Note that the overall probability is $u_1\frac{s-1}{s}$, where u_1 is defined by Eq. (4). Understandably, $u_1 \in [0, 0.5]$ and $u_2 \in [0.5, 1]$ for any $f_{\sum}(\mathcal{C}^i) \geq 0$. For now, we assume we find a certain T such that $u_1 > 0$ holds. The role of T and u_1 will be analyzed later.

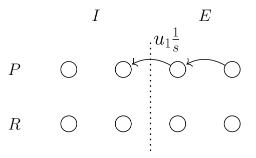
We now consider Sub-case 1 (b). The literal $\neg x_1$ is still included, and we study the transition of TA_3^3 when its current action is "Exclude". The possible transitions are listed below.

Condition: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA_4^3=E.
Therefore, Type II, $x_2 = 1$, $C_3 = \neg x_1 = 0$.

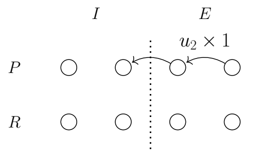


No transition

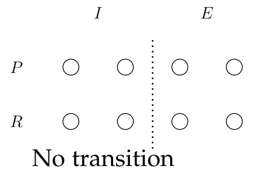
Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA_4^3=E.
Therefore, Type I, $x_2 = 1$, $C_3 = \neg x_1 = 1$.



Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA_4^3=E.
Therefore, Type II, $x_2 = 0$, $C_3 = \neg x_1 = 1$.

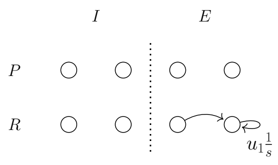


Condition: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA_4^3=I.
Therefore, Type II, $x_2 = 1$, $C_3 = \neg x_1 \wedge \neg x_2 = 0$.

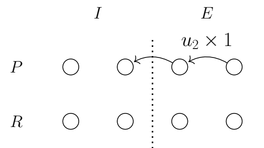


No transition

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA_4^3=I.
Therefore, Type I, $x_2 = 1$, $C_3 = \neg x_1 \wedge \neg x_2 = 0$.

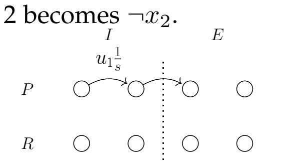


Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA$_4^3$=I.
Therefore, Type II, $x_2 = 0$, $C_3 = \neg x_1 \wedge \neg x_2 = 1$.

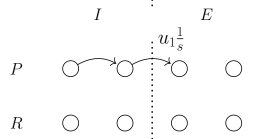


Now let us move onto the third sub-case in Case 1, i.e., Sub-case 1 (c). The literal $\neg x_1$ is still included, and we study the transition of TA$_4^3$ when its current action is "Include". To save space, we remove the instances where no transition happens in the remainder of the paper. Note that we are now studying TA$_4^3$ that corresponds to $\neg x_2$ rather than x_2. Therefore, the literal in Tables 1 and 2 becomes $\neg x_2$.

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = \neg x_1 \wedge \neg x_2 = 0$.

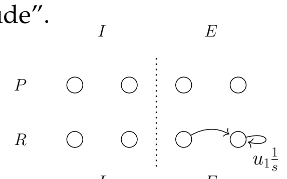


Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

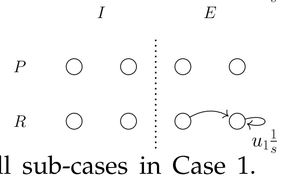


For the Sub-case 1 (d), we study the transition of TA$_4^3$ when it has the current action "Exclude".

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = \neg x_1 = 1$.



Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = \neg x_1 \wedge x_2 = 1$.



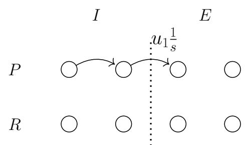
So far, we have gone through all sub-cases in Case 1. We are now ready to sum up Case 1 by looking at the transitions of TA$_3^3$ and TA$_4^3$ in different scenarios. In this case, TA$_4^3$ will become "*Exclude*" in the long run because it has only one direction of transition, i.e., towards action "Exclude". Given TA$_4^3$ is "Exclude", action "Include" of TA$_3^3$ is an absorbing state. Therefore, if TA$_1^3$ and TA$_2^3$ are "Exclude" and "Include", respectively, TA$_3^3$ will become "Include", and TA$_4^3$ will eventually be "Exclude". In other words, C_3 will converge to $\neg x_1 \wedge x_2$ in Case 1.

Case 2

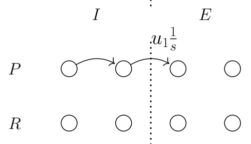
Case 2 studies the behavior of TA$_3^3$ and TA$_4^3$ when TA$_1^3$ and TA$_2^3$ select "Include" and "Exclude", respectively. In this case, the first bit is in the form of x_1 always. There are here also four sub-cases and we will detail them presently.

We first study TA$_3^3$ with action "Include", providing the below transitions.

Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_4^3$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

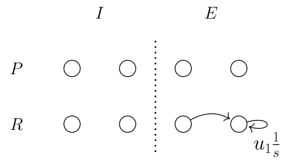


Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_4^3$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

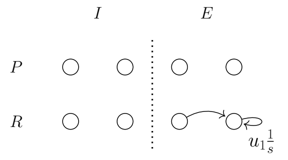


We then study TA$_3^3$ with action "Exclude", and transitions are shown below.

Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_4^3$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

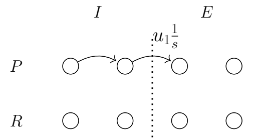


Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_4^3$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

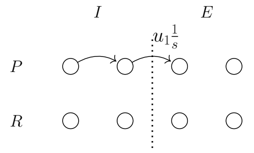


We now study TA$_4^3$ with action "Include" and the transitions are presented below.

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = x_1 \wedge \neg x_2 = 0$.

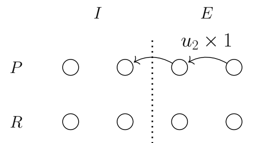


Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

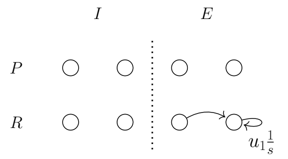


We study lastly TA$_4^3$ with action "Exclude", leading to the following transitions.

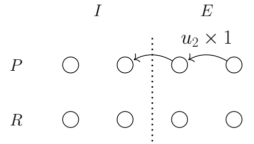
Conditions: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA$_3^3$=E.
Therefore, Type II, $\neg x_2 = 0$, $C_3 = x_1 = 1$.



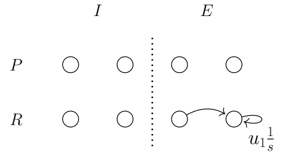
Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.



Conditions: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA$_3^3$=I.
Therefore, Type II, $\neg x_2 = 0$, $C_3 = x_1 \wedge x_2 = 1$.



Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_3^3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = x_1 \wedge x_2 = 0$.



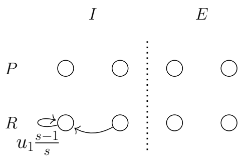
To sum up Case 2, we understand that TA$_3^3$ will select "Exclude", and TA$_4^3$ will switch between "Include" or "Exclude", depending on the training samples and system status.

Case 3

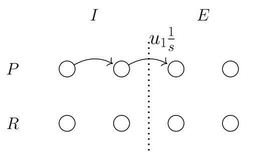
Now we move onto Case 3, where TA$_1^3$ and TA$_2^3$ both select "Exclude". We study the behavior of TA$_3^3$ and TA$_4^3$ for different sub-cases. In this case, the first bit x_1 does not play any role for the output.

We first examine TA$_3^3$ with action "Include", providing the transitions below.

Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$_4^3$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = x_2 = 1$.

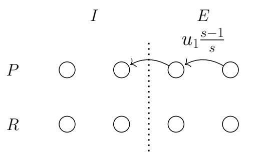


Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

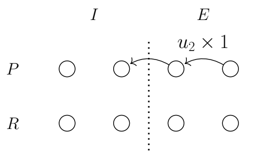


We then study TA$^3_3$ with action "Exclude", transitions shown below. In this situation, if TA$^3_4$ is also excluded, C_3 is "empty" since all its associated TA select action "Exclude". To make the training proceed, according to the training rule of TM, we assign $C_3 = 1$ in this situation.

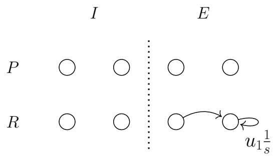
Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = 1$.



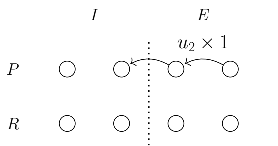
Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA$^3_4$=E.
Therefore, Type II, $x_2 = 0$, $C_3 = 1$.



Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = \neg x_2 = 0$.

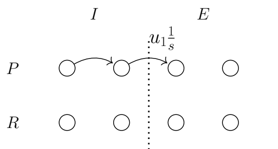


Condition: $x_1 = 0$, $x_2 = 0$, $y = 0$, TA$^3_4$=I.
Therefore, Type II, $x_2 = 0$, $C_3 = \neg x_2 = 1$.

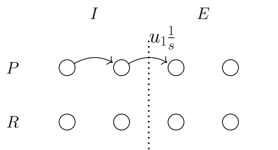


We thirdly study TA$^3_4$ with action "Include", covering the transitions shown below.

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$ TA$^3_3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

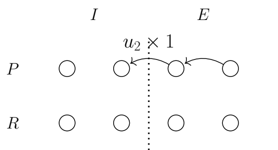


Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

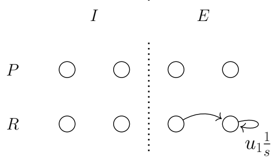


Lastly, we study TA$^3_4$ with action "Exclude", transitions shown below. Similarly, in this situation, when TA$^3_3$ is also excluded, C_3 becomes "empty" again, as all its associated TAs select action "Exclude". Following the training rule of TM, we assign $C_3 = 1$.

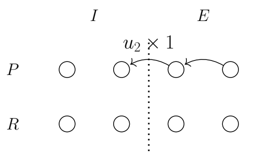
Conditions: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA$^3_3$=E.
Therefore, Type II, $\neg x_2 = 0$, $C_3 = 1$.



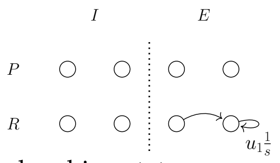
Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 1$.



Conditions: $x_1 = 1$, $x_2 = 1$, $y = 0$, TA$^3_3$=I.
Therefore, Type II, $\neg x_2 = 0$, $C_3 = 1$.



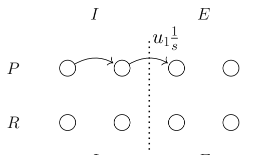
Conditions: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 1$.



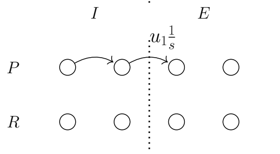
Obviously, in Case 3, there is no absorbing state.

Case 4
Now, we study Case 4, where $\neg x_1$ and x_1 both select "Include". For this reason, in this case, we always have $C_3 = 0$. We study firstly TA$^3_3$ with action "Include" and the transitions are shown below.

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

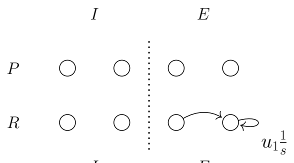


Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

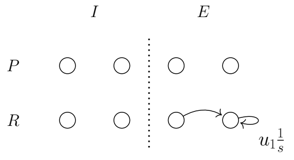


We secondly study TA$^3_3$ with action "Exclude".

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=E.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

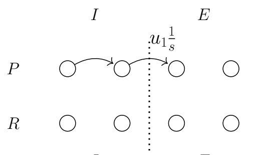


Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_4$=I.
Therefore, Type I, $x_2 = 1$, $C_3 = 0$.

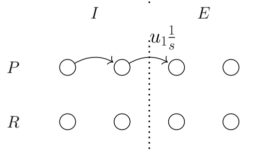


Now, we study TA$^3_4$ with action "Include".

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

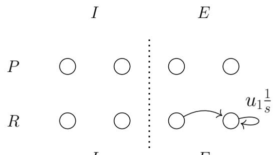


Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.

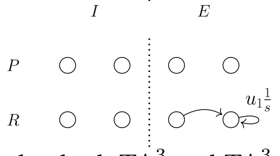


We lastly study TA$^3_4$ with action "Exclude".

Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=E.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.



Condition: $x_1 = 0$, $x_2 = 1$, $y = 1$, TA$^3_3$=I.
Therefore, Type I, $\neg x_2 = 0$, $C_3 = 0$.



To summarize Case 4, we realize that both TA$^3_3$ and TA$^3_4$ will converge to "Exclude".

Based on the above analyses, we can summarize the transitions of TA$^3_3$ and TA$^3_4$, given different configurations of TA$^3_1$ and TA$^3_2$ in Case 1 – Case 4 (i.e., given four different combinations of x_1 and $\neg x_1$). The arrow shown below means the direction of transitions.

Scenario 1: Study TA$^3_3$ = I and TA$^3_4$ = I.

Case 1: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 2: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 3: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 4: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

From the facts presented above, it is confirmed that regardless of the state of TA_1^3 and TA_2^3, if TA_3^3=I and TA_4^3=I, they (TA_3^3 and TA_4^3) will move towards the opposite half of the state space (i.e., towards "Exclude"), away from the current state. So, the state with TA_3^3=I and TA_4^3=I is not absorbing.

Scenario 2: Study TA_3^3 = I and TA_4^3= E.

Case 1: we can see that
$\text{TA}_3^3 \rightarrow \text{I}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 2: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{I, E}$

Case 3: we can see that
$\text{TA}_3^3 \rightarrow \text{I}$
$\text{TA}_4^3 \rightarrow \text{I, E}$

Case 4: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

In this scenario, the starting point of TA_3^3 is "Include" and that of TA_4^3 is "Exclude". In Case 1, where TA_1^3 = E and TA_2^3 = I, TA_3^3 will move towards "Include" and TA_4^3 will move towards "Exclude". Therefore, given TA_1^3 = E and TA_2^3 = I hold, TA_3^3 in "Include" and TA_4^3 in "Exclude" are absorbing actions, while in other cases (i.e., in other configurations of TA_1^3 and TA_2^3), actions "Include" and "Exclude" for TA_3^3 and TA_4^3 are not absorbing.

Scenario 3: Study TA_3^3 = E and TA_4^3 = I.

Case 1: we can see that
$\text{TA}_3^3 \rightarrow \text{I, E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 2: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 3: we can see that
$\text{TA}_3^3 \rightarrow \text{I, E}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 4: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

From the transitions of TA_3^3 and TA_4^3 in Scenario 3, we can conclude that the state with TA_3^3 = E and TA_4^3 = I is not absorbing.

Scenario 4: Study TA_3^3 = E and TA_4^3 = E.

Case 1: we can see that
$\text{TA}_3^3 \rightarrow \text{I}$
$\text{TA}_4^3 \rightarrow \text{E}$

Case 2: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{I, E}$

Case 3: we can see that
$\text{TA}_3^3 \rightarrow \text{I}$
$\text{TA}_4^3 \rightarrow \text{I, E}$

Case 4: we can see that
$\text{TA}_3^3 \rightarrow \text{E}$
$\text{TA}_4^3 \rightarrow \text{E}$

From the transitions of TA_3^3 and TA_4^3 in Scenario 4, we can conclude that the state with TA_3^3 = E and TA_4^3 = E is also absorbing in Case 4, when TA_1^3 and TA_2^3 have both actions as Include.

From the above analysis, we can conclude that when we freeze TA_1^3 and TA_2^3 with certain actions, there are altogether two absorbing cases. (1) Given that TA_1^3 selects "Exclude" and TA_2^3 selects "Include", TA_3^3 selects "Include" and TA_4^3 selects "Exclude". (2) Given that TA_1^3 selects "Include" and TA_2^3 selects "Include", TA_3^3 selects "Exclude" and TA_4^3 selects "Exclude".

So far, we have finished half of the proof. More specifically, we have studied the case when we freeze the transitions of TAs for the first input bit (TA_1^3 and TA_2^3) and examine the transitions of TAs for the second input bit (study TA_3^3 and TA_4^3 by frozen one of them and illustrate the transitions of the other one). In the following paragraphs, we will move on to the second half, i.e., we freeze TA_4^3 and TA_4^3 and study the transition of TA_1^3 and TA_2^3. The analysis procedure is similar to the one that has been done in the above paragraphs, as seen in the following.

1) We freeze TA_3^3 and TA_4^3 as "Exclude" and "Include". In this case, the second bit becomes $\neg x_2$. There are four sub-cases for TA_1^3 and TA_2^3.
   - We study the transition of TA_1^3 when it has the action "Include" as its current action, given different input training samples shown in Table 4 and different actions of TA_2^3 (i.e., when the action of TA_2^3 is frozen as "Include" or "Exclude").
   - We study the transition of TA_1^3 when it has the action "Exclude" as its current action, given different input training samples shown in Table 4 and different actions of TA_2^3 (i.e., when the action of TA_2^3 is frozen as "Include" or "Exclude").
   - We study the transition of TA_2^3 when it has the action "Include" as its current action, given different input training samples shown in Table 4 and different actions of TA_1^3 (i.e., when the action of TA_1^3 is frozen as "Include" or "Exclude").
   - We study the transition of TA_2^3 when it has the action "Exclude" as its current action, given different input training samples shown in Table 4 and different actions of TA_1^3 (i.e., when the action of TA_1^3 is frozen as "Include" or "Exclude").
2) We freeze TA_3^3 and TA_4^3 as "Include" and "Exclude". In this case, the second bit becomes x_2. The sub-cases for TA_1^3 and TA_2^3 are identical to the sub-cases in the previous case.
3) We freeze TA_3^3 and TA_3^3 as "Exclude" and "Exclude". In this case, the second bit is excluded and will not influence the output. The sub-cases for TA_1^3 and TA_2^3 are identical to the sub-cases in the previous case.
4) We freeze TA_3^3 and TA_4^3 as "Include" and "Include". In this case, the second bit will always be 0 and therefore $C_3 = 0$. The sub-cases for TA_1^3 and TA_2^3 are identical to the sub-cases in the previous case.

When we go through all the possible transitions, we can conclude that (1) when TA_3^3 and TA_4^3 are frozen as "Include" and "Exclude", TA_1^3=E and TA_2^3=I are absorbing. (2) When TA_3^3 and TA_4^3 are frozen as "Include" and "Include", TA_1^3=E and TA_2^3=E are also absorbing. The detailed proof of this statement can be found in Appendix 2.

When we look at the absorbing cases that are conditioned upon the frozen actions of both "Include" (i.e., when TA_3^3 and TA_4^3 are frozen as Include and Include, and when TA_1^3 and TA_2^3 are frozen as Include and Include), we can easily conclude that those conditions cannot be fulfilled and thus the corresponding system states are not absorbing.

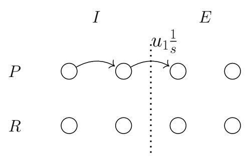
The main reason is that the condition of both "Include" is surely not absorbing and such state cannot be frozen as a stable state. Differently, for the state with TA$_1^3$=E, TA$_2^3$=I, TA$_3^3$=I and TA$_4^3$=E, the learning mechanism together with the training samples will reinforce the individual TA to move to a deeper state for the selected actions. Therefore, the state is indeed an absorbing state and it is the only absorbing state in the system. Therefore, given infinite time horizon, the TM will converge to the expected logic, which is half of the XOR-relation. We thus prove Lemma 1. $\square$

Following the same strategy used for the proof in Lemma 1, we can prove Lemma 2. We do not detail the proof for the sake of brevity.

3.2.2 Proof of Lemma 3

Proof. To prove that the system is recurrent, we just need to show that the only absorbing state in the TM based on the training samples from Table 5 disappears when the training samples in Table 3 is given. More specifically, once the TM is trained based on Table 5, we will show that the absorbing state disappears when training sample $x_1 = 1$, $x_2 = 0$, and $y = 1$ is given in addition. To validate this point, we can simply show that one of the TA, i.e., TA$_3^3$ with an "Including" action will not move only towards "Include" when the output of the first literal is $\neg x_1$, and when TA$_4^3$ is "Exclude", given the newly added training sample. The diagram below indicates the transition of TA$_3^3$ in the above mentioned condition when the new training sample is given.

Condition: $x_1 = 1$, $x_2 = 0$, $y = 1$, TA$_4^3$=E.
Therefore, we have Type I feedback for literal $x_2 = 0$, clause $C_3 = \neg x_1 \wedge x_2 = 0$.



When $x_1 = 1$, $x_2 = 0$, and $y = 1$ is given in addition, TA$_3^3$ has a non-zero probability to move towards "Exclude". Therefore, "Include" is not the only direction that TA$_3^3$ moves to upon the input, and this will make the state not absorbing any longer. For other states, the newly added training sample will not remove any transition from the previous case. Therefore, the system will not have any new absorbing state. Given the non-zero probability of returning II, IE, EI, EE, these system states are recurrent. $\square$

To summarize so far, from Lemma 1 and Lemma 2, we understand that each individual clause is able to learn any sub-pattern from the XOR-relation. However, when the full logic of XOR is given, as shown in Table 3, the system state II, IE, EI and EE becomes recurrent. In other words, each clause may stay in any of the above states in probability that is less than 1. For this reason, it is necessary to have a parameter to guide the learning process of different clauses so that they can converge or cover different sub-patterns. The parameter is T, and the analysis is given presently.

3.2.3 Proof of Lemma 4 and Lemma 5

Proof. Consider m clauses in the TM, and $m > T$. Let's study parameter u_1 first. When $f_{\sum}(\mathcal{C}_i)$ is zero, $u_1 = 1/2$. When $0 < f_{\sum}(\mathcal{C}_i) < T$, $u_1 = \frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}_i)))}{2T} = \frac{T - f_{\sum}(\mathcal{C}_i)}{2T}$, and u_1 monotonically decreases as $f_{\sum}(\mathcal{C}_i)$ increases. When $T \geq m$, $u_1 = 0$ holds. For u_2, when $f_{\sum}(\mathcal{C}_i) = 0$, $u_2 = 1/2$. When $0 < f_{\sum}(\mathcal{C}_i) < T$, $u_2 = \frac{T + f_{\sum}(\mathcal{C}_i)}{2T}$, and it monotonically increases when $f_{\sum}(\mathcal{C}_i)$ grows. When $T \geq m$, $u_2 = 1$ holds.

Clearly, $u_2 > 0$ always holds regardless the value of $f_{\sum}(\mathcal{C}_i)$. Therefore, the variations of $f_{\sum}(\mathcal{C}_i)$ does not change the directions of system transitions upon Type II feedback.

To guarantee $u_1 > 0$, it is required $0 \leq f_{\sum}(\mathcal{C}_i) < T$. When this condition fulfills, the variations of $f_{\sum}(\mathcal{C}_i)$ does not change the directions of system transitions upon Type I feedback.

According to the system updating rule of TM, once the Type I or Type II feedback is triggered, the clauses are updated independently. Due to the recurrent property, each clause will transit among II, IE, EI and EE, as long as $0 \leq f_{\sum}(\mathcal{C}_i) < T$. In other words, each clause will move among those four status, until T clauses follow the same sub-pattern. Therefore, consider infinite time horizon, the event that $T, T < m$, clauses appear in the same sub-pattern will almost surely happen.

$\square$

When $f_{\sum}(\mathcal{C}_i) = T$, it means there are T clauses that have followed the same sub-pattern. Once this happens, it means that there are certain number of clauses that have learnt a certain sub-pattern already and we would like to encourage the other clauses to learn the other sub-pattern. This is to be shown in Lemma 5.

Proof. Lemma 5 is self-evident. When $f_{\sum}(\mathcal{C}_i) = T$, $u_1 = 0$ holds and thus Type I feedback will not be generated to the TM for any updates when the same training sample is given. For example, without loss of generalization, we assume there are T clauses that have converged to $c_3 = \neg x_1 \wedge x_2 = 1$. When another training sample $x_1 = 0$, $x_2 = 1$, $y = 1$ is given, Type I feedback will not be given any longer because $u_1 = 0$. Therefore, such input training sample is filtered out and the system will only update for Type I feedback when training sample $x_1 = 1$, $x_2 = 0$, $y = 1$ is given (i.e., the TM will update based on the samples shown in Table 5, guiding the TM to learn the other sub-pattern according to Lemma 2.) $\square$

3.2.4 Proof of Theorem 2

Proof. Based on Lemmas 1-5, we can prove Theorem 2.

Clearly, u_1 monotonically decreases as $f_{\sum}(\mathcal{C}_i)$ increases. When the number of clauses that follow a certain sub-pattern increases, due to the monotonicity of u_1, the impact of such training samples becomes less and less to the system. Ultimately, when $f_{\sum}(\mathcal{C}_i) = T$ holds, the system will not be updated for the learnt sub-pattern. Therefore, at this particular time, only the other sub-pattern will be used for system training. This behavior can avoid the situation that many clauses learn one sub-pattern but the other sub-pattern is not learnt.

Now let's consider the case where the selected T is less than or equal to half of the number of the clauses, i.e., $T \leq m/2$. From Lemma 4, we know that the system will eventually have T clauses that follow one sub-pattern. Once this happens, due to Lemmas 1 and 2, we understand that all clauses will move towards the only absorbing state of the system corresponds to the other sub-pattern. As soon as the number of the clauses that follow each sub-pattern reaches T, the system will not be updated any longer for any training input. In this situation, the system have been absorbed to the point where both sub-patterns of XOR have been learnt.

We thus complete the proof. □

Note that the clauses that already follow a sub-pattern may get out of the sub-pattern when training samples from the other sub-pattern are given. So even if there are T clauses that are have learnt a sub-pattern, the number of learnt clauses may decrease in front of training samples from the other sub-pattern. However, as soon as the sum of the clauses for the same sub-pattern is less than T, the corresponding Type I feedback can be triggered again for this sub-pattern, leading the clauses to possibly move back to the sub-pattern again. Nevertheless, when the number of the clauses that follow each sub-pattern reaches T, the system is then converged to and locked at the intended pattern.

Note also that when T is less than half of the number of the clauses, there are $m - 2T$ clauses that do not follow to any of the sub-patterns when the system stops updating. Therefore, those clauses that do not follow the correct XOR sub-patterns may involve incorrect output if they all happen to follow a certain incorrect logic and the sum of them happens to be greater than or equal to T. For this reason, even if the absorbing states exist, the number of clauses, the threshold value T need to be carefully chosen.

Remark 1. *The system configuration described in Theorem 1 is a special case of Theorem 2.*

Remark 2. *When T is greater than half of the number of the clauses, i.e., $T > m/2$, the system will not have any absorbing state. We conjuncture that the system can still learn the two sub-patterns in a balanced manner, as long as T is not configured too close to the total number of clauses m and when s is sufficiently large.*

To address the conjecture in Remark 2, we now study the system behavior when $T > m/2$. According to Lemma 3, at a certain time slot, there will be T clauses following a certain sub-pattern, named sub-pattern 1. In this situation, the corresponding training samples for sub-pattern 1 will not trigger any Type I feedback to the system and therefore the other training samples will guide the remaining clauses to learn towards the other sub-pattern, named sub-pattern 2. As the training process continues, all the clauses (including those T clauses who have learned sub-pattern 1, although the action probability (u_1/s) is low) will lean towards sub-pattern 2.

Because $m - T < T$, the clauses following sub-pattern 2 will not block the training samples for sub-pattern 2 even if there are $m - T$ clauses that follow this sub-pattern. Therefore, as more training samples are given, the remaining clauses will eventually move out of sub-pattern 1 or their current states and then move towards sub-pattern 2, until T clauses follow sub-pattern 2 before the training samples for the sub-pattern 2 are completely blocked. Then the clauses will again move towards sub-pattern 1.

The system will thus oscillate and will not be absorbed to a certain state. Nevertheless, with high probability, the system will have at least $m - T$ clauses that follow each sub-pattern, especially when s is large. According to the updating rule of Type I feedback, the probability for an included literal in a clause that has learnt a certain sub-pattern to change towards the other sub-pattern is u_1/s, which only happens when a training sample of the other sub-pattern is given. On the other hand, the reward is $u_1\frac{s-1}{s}$ if a training sample of the same sub-pattern is received. Therefore, when s is large, the clause is less likely to get out of the learnt sub-pattern due to a training sample from the conflicting sub-pattern. In other words, the system will most probably have at least $m - T$ clauses for each sub-pattern after training, and in the worst case, $m - 2(m - T)$ clauses will appear in states other than any intended sub-pattern, depending on the stop time of training. To summarize, if we select the threshold as $m - T$, the two sub-patterns of the XOR-relation can still be followed with high probability.

4 Simulation experiment

To validate the above theoretical results, we have added two experiments[3]:

- We employ 5 clauses and set $T = 2$. Based on the analytical result, i.e., Theorem 2, the system will be absorbed at one state, where two clauses learn one sub-pattern, two other clauses learn the other sub-pattern, and the remaining clause is in any state that may or may not belong to any one of the sub-patterns. Once the system is absorbed, no update will happen in TAs for any input training samples.
- We employ 5 clauses and set $T = 3$. In this case, as stated in Remark 2, the system does not have an absorbing state. Instead, four clauses will still cover all sub-patterns, two for each sub-pattern, with high probability. The remaining one transits back and forth between the sub-patterns. In this case, the TAs will still update their states upon new input samples.

Figures 5 and 6 show the output of different clauses as a function of the training epochs for $T = 2$ and $T = 3$ respectively. In those figures, when a clause recognizes the sub-pattern ($x_1 = 1$, $x_2 = 0$), it is represented by a blue star. Similarly, when a clause learns the sub-pattern ($x_1 = 0$, $x_2 = 1$), it is represented by a red square. All the other possible states are represented with a black triangular. Figure 7 shows the variation of the number of TA updates per epoch for the above different T values.

From Figure 5, we can see that the clauses have random initial starting points. When the input samples are given, the states become updated. At around epoch 60, two clauses learn one intended sub-pattern (in red block) and two clauses learn the other sub-pattern (in blue star), and Clause 3 is in another state which is different from any of the sub-patterns of XOR. Thereafter, the system will not update any longer, which can be confirmed by the curve of $T = 2$ after epoch 60 in Figure 7. From Figure 6, we can see that the system will not be absorbed for $T = 3$ (also see the curve

3. The simulation code for XOR-relation can be found at https://github.com/cair/TM-XOR-proof.

for $T = 3$ in Figure 7), but each sub-pattern of XOR has been covered by at least two clauses after about epoch 60. Not surprisingly, there are indeed a few observed cases for $T = 2$ and $T = 3$ that two clauses have followed the two distinct sub-patterns respectively and one clause is in a non-intended pattern. Nevertheless, due to the majority voting mechanism of TM, the trained TM can still give the correct output of the XOR-relation.

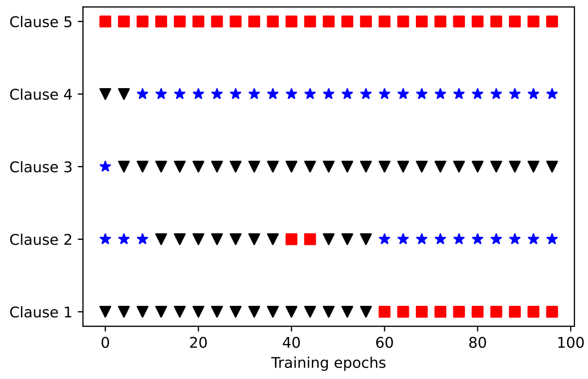


Fig. 5: The convergence of a TM with 5 clauses when $T = 2$.

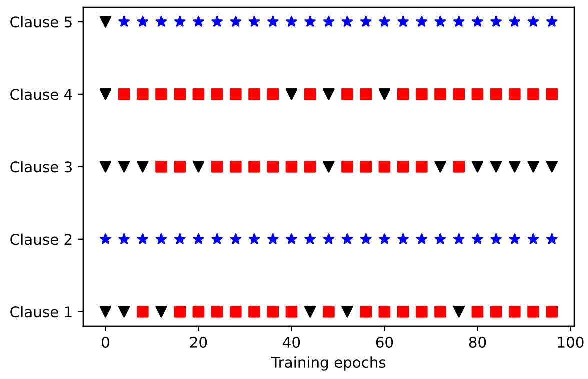


Fig. 6: The convergence of a TM with 5 clauses when $T = 3$.

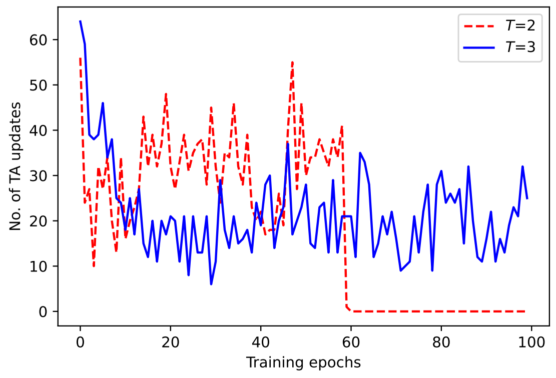


Fig. 7: The number of TA updates as a function of training epochs.

5 Conclusions

In this paper, we complete the proof on the convergence of the XOR-relation. Firstly, we demonstrate that TM can almost surely learn the XOR-relation with the simplest structure. Thereafter, we analyze the dynamics of the system and reveal the relationship between the number of clauses and the hyper-parameter T when multiple sub-patterns exist. The analytical results not only confirm the convergence property of TM in XOR-relation, they also illustrate the role of the hyper-parameter T when multiple sub-patterns exist.

Acknowledgment

This work was supported by the project Spacetime Vision: Towards Unsupervised Learning in the 4D World financed by the EEA and Norway Grants 2014-2021 under the grant number EEA-RO-NO-2018-04. This work was also supported by the Research Council of Norway through the AI4Citizen project (320783) and the AIEverywhere project (312434).

References

[1] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv:1804.01508*, Apr. 2018.
[2] 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 *IEEE ICECS*, 2020.
[3] R. Saha, O.-C. Granmo, V. I. Zadorozhny, and M. Goodwin, "A Relational Tsetlin Machine with Applications to Natural Language Understanding," *Journal of Intelligent Information Systems*, pp. 1–28, 2022.
[4] M. L. Tsetlin, "On Behaviour of Finite Automata in Random Medium," *Avtomat. i Telemekh*, vol. 22, no. 10, pp. 1345–1354, 1961.
[5] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," in *ACM SIGKDD*, 2016.
[6] C. Rudin, "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.
[7] L. G. Valiant, "A Theory of the Learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[8] C. D. Blakely and O.-C. Granmo, "Closed-Form Expressions for Global and Local Interpretation of Tsetlin Machines with Applications to Explaining High-Dimensional Data," *arXiv preprint arXiv:2007.13885*, 2020.
[9] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, "Learning Automata based Energy-efficient AI Hardware Design for IoT," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2182, 2020.
[10] R. Shafik, A. Wheeldon, and A. Yakovlev, "Explainability and Dependability Analysis of Learning Automata based AI Hardware," in *IEEE IOLTS*, 2020.
[11] O.-C. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The Convolutional Tsetlin Machine," *arXiv preprint arXiv:1905.09688*, 2019.
[12] K. Darshana Abeyrathna, O.-C. Granmo, X. Zhang, L. Jiao, and M. Goodwin, "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, 2020.
[13] K. D. Abeyrathna, O.-C. Granmo, and M. Goodwin, "A Regression Tsetlin Machine with Integer Weighted Clauses for Compact Pattern Representation,," in *IEA/AIE*, 2020.
[14] A. Phoulady, O. C. Granmo, S. Gorji, and H. A. Phoulady, "The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting," in *Ninth International Workshop on Statistical Relational AI (StarAI 2020)*, 2020.
[15] S. Gorji, O. C. Granmo, S. Glimsdal, J. Edwards, and M. Goodwin, "Increasing the Inference and Learning Speed of Tsetlin Machines with Clause Indexing," in *IEA/AIE*, 2020.

[16] K. D. Abeyrathna, O.-C. Granmo, and M. Goodwin, "Extending the Tsetlin Machine With Integer-Weighted Clauses for Increased Interpretability," *IEEE Access*, vol. 9, pp. 8233–8248, 2021.

[17] B. J. Oommen, "Stochastic searching on the line and its applications to parameter learning in nonlinear optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 4, pp. 733–739, 1997.

[18] J. Sharma, R. K. Yadav, O. C. Granmo, and L. Jiao, "Drop Clause: Enhancing Performance, Interpretability and Robustness of the Tsetlin Machine," *arXiv preprint arXiv:2105.14506*, 2022.

[19] 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. 115 134–115 146, 2019.

[20] B. Bhattarai, O. C. Granmo, and L. Jiao, "ConvTextTM: An Explainable Convolutional Tsetlin Machine Framework for Text Classification," in *LREC*, 2022.

[21] R. K. Yadav, L. Jiao, O. C. Granmo, and M. Goodwin, "Interpretability in Word Sense Disambiguation using Tsetlin Machine," in *ICAART*, 2021.

[22] B. Bhattarai, L. Jiao, and O. C. Granmo, "Measuring the Novelty of Natural Language Text Using the Conjunctive Clauses of a Tsetlin Machine Text Classifier," in *ICAART*, 2021.

[23] B. Bhattarai, O. C. Granmo, and L. Jiao, "Word-level human interpretable scoring mechanism for novel text detection using Tsetlin Machines," *Applied Intelligence*, 2022.

[24] ——, "Explainable Tsetlin Machine framework for fake news detection with credibility score assessment," in *LREC*, 2022.

[25] R. Saha, O. C. Granmo, and M. Goodwin, "Using Tsetlin Machine to Discover Interpretable Rules in Natural Language Processing Applications," *Expert Systems*, 2021.

[26] R. K. Yadav, L. Jiao, O. C. Granmo, and M. Goodwin, "Human-Level Interpretable Learning for Aspect-Based Sentiment Analysis," in *AAAI*, 2021.

[27] ——, "Robust Interpretable Text Classification against Spurious Correlations Using AND-rules with Negation," in *IJCAI*, 2022.

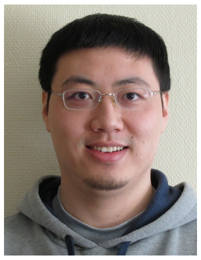
[28] S. Glimsdal and O. C. Granmo, "Coalesced Multi-Output Tsetlin Machines with Clause Sharing," *arXiv preprint arXiv:2108.07594*, 2021.

[29] X. Zhang, L. Jiao, O.-C. Granmo, and M. Goodwin, "On the Convergence of Tsetlin Machines for the IDENTITY-and NOT Operators," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

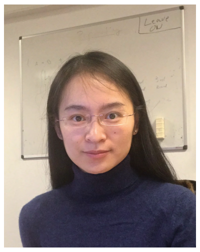
[30] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Prentice-Hall, Inc., 1989.

[31] X. Zhang, L. Jiao, B. J. Oommen, and O.-C. Granmo, "A Conclusive Analysis of the Finite-time Behavior of the Discretized Pursuit Learning Automaton," *IEEE Trans. Neural Netw. Learn. Sys.*, vol. 31, no. 1, pp. 284–294, 2020.

[32] M. Minsky and S. Papert, "Perceptrons." 1969.



Lei Jiao (M'12-SM'18) received his BE degree from Hunan University, China, in 2005. He received his ME degree from Shandong University, China, in 2008. He obtained his PhD degree in Information and Communications Technology from University of Agder (UiA), Norway, in 2012. He is now an Associated Professor in the Department of Information and Communication Technology, UiA. His research interests include reinforcement learning, learning automata, natural language processing, resource allocation and performance evaluation for communication and energy systems.



Xuan Zhang received her PhD degree in Information and Communication Technology from the University of Agder (UiA) in 2015. She has a Master's degree in Signal and Information Processing and a Bachelor's degree in Electronics and Information Engineering. She is now a senior researcher in Norwegian Research Center (NORCE). At the same time, she is a scientific researcher at Centre of Artificial Intelligence Research (CAIR) in UiA. She is currently serving as a Board Member of Norwegian Association for Image Processing and Machine Learning. Her research interests include: Learning Automata, Mathematical Analysis on Learning Algorithms, Stochastic Modeling and Optimization, Deep Learning, Natural Language Processing and Computer Vision.

Ole-Christoffer Granmo is a Professor and Founding Director of Centre for Artificial Intelligence Research (CAIR), University of Agder, Norway. He obtained his master's degree in 1999 and the PhD degree in 2004, both from the University of Oslo, Norway. Dr. Granmo has authored in excess of 140 refereed papers with numerous best paper awards, encompassing learning automata, bandit algorithms, Tsetlin machines, Bayesian reasoning, reinforcement learning, and computational linguistics. He has further coordinated 7+ Norwegian Research Council projects and graduated more than 60 master- and PhD students. Dr. Granmo is also a co-founder of the Norwegian Artificial Intelligence Consortium (NORA). Apart from his academic endeavours, he co-founded Anzyz Technologies AS..

Kuruge Darshana Abeyrathna completed his BSc in Mechatronics Engineering at AIT university, Thailand in 2015. Then he directly joined the Big Data research group at Thammasat University, Thailand for his MSc studies and graduated in 2017. Darshana completed his PhD in Computer Science, specializing in Machine Learning, from the University of Agder, Norway, in 2022. His research interests are in Tsetlin Machine, Artificial Neural Networks, Data Mining, Optimization, and Operations Research. Currently, at DNV, he is working as a senior researcher.