The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic*

Ole-Christoffer Granmo†

Abstract

Although simple individually, artificial neurons provide state-of-the-art performance when interconnected in deep networks. Arguably, the Tsetlin Automaton is an even simpler and more versatile learning mechanism, capable of solving the multi-armed bandit problem. Merely by means of a single integer as memory, it learns the optimal action in stochastic environments through increment and decrement operations. In this paper, we introduce the Tsetlin Machine, which solves complex pattern recognition problems with propositional formulas, composed by a collective of Tsetlin Automata. To eliminate the longstanding problem of vanishing signal-to-noise ratio, the Tsetlin Machine orchestrates the automata using a novel game. Further, both inputs, patterns, and outputs are expressed as bits, while recognition and learning rely on bit manipulation, simplifying computation. Our theoretical analysis establishes that the Nash equilibria of the game align with the propositional formulas that provide optimal pattern recognition accuracy. This translates to learning without local optima, only global ones. In five benchmarks, the Tsetlin Machine provides competitive accuracy compared with SVMs, Decision Trees, Random Forests, Naive Bayes Classifier, Logistic Regression, and Neural Networks. We further demonstrate how the propositional formulas facilitate interpretation. In conclusion, we believe the combination of high accuracy, interpretability, and computational simplicity makes the Tsetlin Machine a promising tool for a wide range of domains.

Keywords: Bandit Problem, Game Theory, Interpretable Pattern Recognition, Propositional Logic, Tsetlin Automata Games, Learning Automata, Frequent Pattern Mining, Resource Allocation.

1   Introduction

Although simple individually, artificial neurons provide state-of-the-art performance when interconnected in deep networks [1]. However, deep neural networks often require huge amounts of training data and extensive computational resources. Unknown to many, there exists an arguably even more fundamental and versatile learning mechanism than the artificial neuron, namely, the Tsetlin Automaton, developed by M.L. Tsetlin in the Soviet Union in the early 1960s [2]. In this paper, we propose a novel technique for large-scale and complex pattern recognition based on Tsetlin Automata.

1.1   The Tsetlin Automaton

Tsetlin Automata have been used to model biological systems, and have attracted considerable interest because they can learn the optimal action when operating in unknown stochastic environments [2, 3]. Furthermore, they combine rapid and accurate convergence with low computational complexity.

In all brevity, the Tsetlin Automaton is one of the pioneering solutions to the well-known multi-armed bandit problem [4, 5] and the first Learning Automaton [3]. <mark>It performs actions</mark> α_z, $z \in \{1, 2\}$, sequentially in an environment. Each action α_z triggers either a reward or a penalty. That is, the action is rewarded with probability p_z, otherwise it is penalized. The reward probabilities are unknown to the automaton and may change over time. Under such challenging conditions, the goal is to identify the action with the highest reward probability using as few attempts as possible.
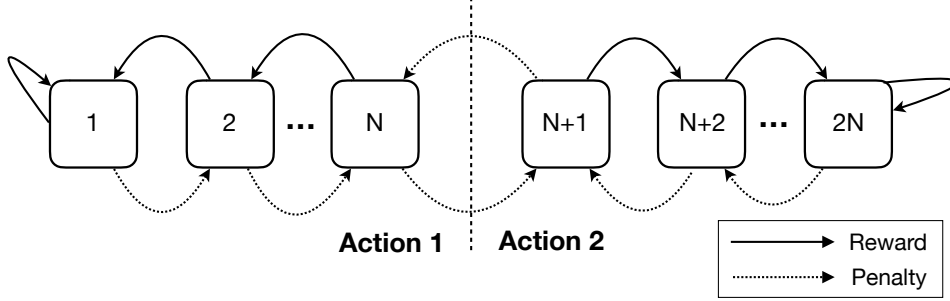

Figure 1: A Tsetlin Automaton for two-action environments.

The mechanism driving the Tsetlin Automaton is surprisingly simple. Informally, as illustrated in Figure 1, a Tsetlin Automaton is simply a fixed finite-state automaton [6] with an unusual interpretation:

- The current state of the automaton decides which action to perform. The automaton in the figure has $2N$ states. Action 1 (α_1) is performed in the states with index 1 to N, while Action 2 (α_2) is performed in the states with index $N + 1$ to $2N$.

- The state transitions of the automaton govern learning. One set of state transitions is activated on reward (solid lines in the figure), and one set of state transitions is activated on penalty (dotted lines in the figure). As seen, rewards and penalties trigger specific transitions from one state to another, designed to reinforce successful actions (those eliciting rewards).

Formally, a Two-Action Tsetlin Automaton can be defined as a quintuple [3]:

$$\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, F(\cdot, \cdot), G(\cdot)\}.$$

$\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_{2N}\}$ is the set of internal states. $\underline{\alpha} = \{\alpha_1, \alpha_2\}$ is the set of automaton actions. $\underline{\beta} = \{\beta_{\text{Penalty}}, \beta_{\text{Reward}}\}$ is the set of inputs that can be given to the automaton. An output function $G(\phi_u)$ determines the next action performed by the automaton, given the current automaton state ϕ_u:

$$G(\phi_u) = \begin{cases} \alpha_1, & \textbf{if } 1 \leq u \leq N \\ \alpha_2, & \textbf{if } N + 1 \leq u \leq 2N. \end{cases} \tag{1}$$

Finally, a transition function $F(\phi_u, \beta_v)$ determines the new automaton state from: (i) the current automaton state ϕ_u and (ii) the response β_v of the environment to the action performed by the automaton:

$$F(\phi_u, \beta_v) = \begin{cases} \phi_{u+1}, & \textbf{if } 1 \leq u \leq N \textbf{ and } v = \text{Penalty} \\ \phi_{u-1}, & \textbf{if } N + 1 \leq u \leq 2N \textbf{ and } v = \text{Penalty} \\ \phi_{u-1}, & \textbf{if } 1 < u \leq N \textbf{ and } v = \text{Reward} \\ \phi_{u+1}, & \textbf{if } N + 1 \leq u < 2N \textbf{ and } v = \text{Reward} \\ \phi_u, & \textbf{otherwise}. \end{cases} \tag{2}$$

Implementation-wise, a Tsetlin Automaton simply maintains an integer (the state index), and learning is performed through increment and decrement operations, according to the transitions specified by $F(\phi_u, \beta_v)$ (and depicted in Figure 1). *The Tsetlin Automaton is thus extremely simple computationally, with a very small memory footprint.*

1.2 State-of-the-art in the Field of Learning Automata

Learning Automata have attracted considerable interest because they can learn the optimal action when operating in unknown stochastic environments [3]. As such, they have been used for pattern classification over more than four decades. Early work includes the stochastic Learning Automata-based classifier of Barto and Anandan [7], as well as the games of Learning Automata proposed by Narendra and Thathachar [3]. These approaches can learn the optimal classifier for specific forms of discriminant functions (e.g., linear classifiers), also when feedback is noisy. Along the same lines, Sastry and Thathachar have provided several algorithms based on cooperating systems of Learning Automata [8]. More recently, Zahiri proposed hyperplane- and rule-based classifiers, which performed comparably to other well-known methods in the literature [9, 10]. Recent research also includes the noise-tolerant learning algorithm by Sastry et al., built upon a team of continuous-action Learning Automata [11]. Further, Goodwin et al. proposed a Learning Automata-guided random walk on a grid to construct robust discriminant functions [12], while Motieghader et al. introduced a hybrid scheme that combines a genetic algorithm with Learning Automata for classification [13].

In general, however, previous Learning Automata schemes have mainly addressed relatively small-scale pattern recognition tasks. The above solutions are further primarily based on so-called *variable structure learning automata* [14]. Although still simple, these are significantly more complex than the Tsetlin Automaton because they need to maintain an action probability vector for sampling actions. Their success in pattern recognition is additionally limited by constrained pattern representation capability (linearly separable classes and simple decision trees).

The Tsetlin Automaton has formed the core of more advanced learning automata designs as well. This includes decentralized control [15], searching on the line [16], equi-partitioning [17], streaming sampling for social activity networks [18], faulty dichotomous search [19], learning in deceptive environments [20], and routing in telecommunication networks [21]. The strength of these finite state learning automata solutions is that they have provided state-of-the-art performance when problem properties are unknown and stochastic, while the problem must be solved as quickly as possible through trial and error.

1.3 The Vanishing Signal-to-Noise Ratio Problem

The ability to handle stochastic and unknown environments for a wide range of problems, combined with their computational simplicity and small memory footprint, make Learning Automata an attractive building block for complex machine learning tasks. However, a particularly adverse challenge has hindered the success of Learning Automata. Complex problem solving requires a team of interacting automata, and unfortunately, each team member introduces noise. This challenge manifests sharply in the Goore Game, which employs a simple uni-modal stochastic objective function to be optimized by a Learning Automata team [22, 15].

The challenge manifests as follows. Consider a team of $W + 1$ two-action variable structure Learning Automata. Each automaton maintains an action probability p. In any round of the Goore Game, an automaton performs its first action with probability p and its second action with probability $1 - p$. Now, let μ be the average value with which a single automaton can modify the game's objective function by switching action. From the perspective of a single automaton, each other automaton in the team then increases the objective function variance, amounting to $p(1 - p)\mu^2$ [22]. In particular, consider the case of a deterministic objective function, with p being the action probability within the team that currently is farthest away

from 0.5. Then the best-case signal-to-noise ratio (SNR) becomes:

$$\text{SNR} = \frac{\mu^2}{\sigma^2} = \frac{\mu^2}{Wp(1-p)\mu^2} = \frac{1}{Wp(1-p)}. \tag{3}$$

That is, as the number of automata W grows, SNR drops to zero (vanishes).

For Tsetlin Automata, which are deterministic, the stochasticity of the action selection arises from the automata changing state. These state changes are stochastic since they are driven by stochastic feedback. Thus, again, as the number of automata grows, so does the variance of the feedback. Even for a simple uni-modal objective function, such as the one used in the Goore Game, this effect is problematic. Indeed, as explored by Kleinrock and Tung in 1996, a team of Tsetlin Automata solving the Goore Game needs to maintain an increasingly larger state-space as the number of Tsetlin Automata grows, and in the end an infinite number of states per Tsetlin Automaton [15].

In general, this problem is inherent to decision-making with Learning Automata because of their decentralized and stochastic nature. The automata independently decide upon their actions, directly based on the feedback from the environment. This is on one hand a strength because it allows problems to be solved in a decentralized manner. On the other hand, as the number of automata grows, the level of noise increases. In the following, we will refer to this effect as the *vanishing signal-to-noise ratio problem*.

```
0 0 * 1 * 0 0 0
* 0 * 1 * 0 0 0
0 * * 1 * * * 0
0 * * * * 0 0 *
0 0 0 * * 0 0 0
0 * 0 * * * 0 0
0 0 * 1 * * * 0
0 0 0 * 1 * * *
```

Table 1: A bit pattern produced by the Tsetlin Machine for handwritten digits '1'. The '*' symbol can either take the value '0' or '1'. The remaining bit values require strict matching. The pattern is relatively easy to interpret for humans compared to, e.g., the weights of a neural network. It is also efficient to evaluate for computers. Despite this simplicity, the Tsetlin Machine produces bit patterns that deliver competitive pattern recognition accuracy for several datasets, as reported in Section 5.

1.4 Paper Contributions

In this paper, we attack the limited pattern expression capability and vanishing signal-to-noise ratio of Learning Automata-based pattern recognition, introducing the *Tsetlin Machine*. The contributions of the paper can be summarized as follows:

- We introduce the Tsetlin Machine, which solves complex pattern recognition problems with *propositional formulas*, composed by a collective of Tsetlin Automata.

- We eliminate the longstanding vanishing signal-to-noise ratio problem with a unique decentralized learning scheme based on game theory [23, 2]. The game we have designed allows *millions* of Tsetlin Automata to successfully cooperate.

- The game orchestrated by the Tsetlin Machine is based on resource allocation principles [24], in inter-play with frequent pattern mining [25]. By allocating sparse pattern representation resources according to the frequency of the patterns, the Tsetlin Machine is able to capture intricate unlabelled sub-patterns, for instance addressing the so-called Noisy XOR-problem.

- Our theoretical analysis establishes that the Nash equilibria of the Tsetlin Machine game are aligned with the propositional formulas that provide optimal pattern recognition accuracy. This translates to learning without local optima, only global ones.

- The propositional formulas are represented as bit patterns. These bit patterns are relatively easy to interpret, compared to e.g. a neural network (see Table 1 for an example bit pattern). This facilitates human quality assurance and scrutiny, which for instance can be important in safety-critical domains such as medicine.

- The Tsetlin Machine is a new approach to global construction of decision rules [26, 27]. We demonstrate that decision rules for large-scale pattern recognition can be learned on-line, under particularly noisy conditions.

- The Tsetlin Machine is particularly suited for digital computers, being directly based on bit manipulation with AND-, OR-, and NOT operators.

- In our empirical evaluation on five datasets, the Tsetlin Machine provides competitive performance in comparison with Multilayer Perceptron Networks, Support Vector Machines, Decision Trees, Random Forests, the Naive Bayes Classifier, and Logistic Regression.

- We demonstrate how the Tsetlin Machine can be used as a building block to create more advanced architectures.

In conclusion, we believe the combination of high accuracy, interpretability, and computational simplicity makes the Tsetlin Machine a promising tool for a wide range of domains.

1.5   Paper Organization

The paper is organized as follows. In Section 2, we define the exact nature of the pattern recognition problem we are going to solve, also introducing the crucial concept of sub-patterns.

Then, in Section 3, we cover the Tsetlin Machine in detail. We first present the Tsestlin Machine inference structure, before we introduce the Tsetlin Automata teams that write formulas in propositional logic. These Tsetlin Automata teams are in turn organized to recognize complex patterns. We conclude the section by presenting the Tsetlin Machine game that we use to coordinate millions of Tsetlin Automata, eliminating the vanishing signal-to-noise ratio problem.

In Section 4, we analyze pertinent properties of the Tsetlin Machine formally, and establish that the Nash equilibria of the game are aligned with the propositional formulas that solve the pattern recognition problem at hand. This allows the Tsetlin Machine as a whole to robustly and accurately uncover complex patterns with propositional logic.

In our empirical evaluation in Section 5, we evaluate the performance of the Tsetlin Machine on five datasets: Flower categorization, digit recognition, board game planning, the Noisy XOR Problem with Non-informative Features, as well as MNIST.

The Tsetlin Machine has been designed to act as a building block in more advanced architectures, and in Section 6 we demonstrate how four distinct architectures can be built by interconnecting multiple Tsetlin Machines.

As the first step in a new research direction, the Tsetlin Machine also opens up a range of new research questions. In Section 7, we summarize our main findings and provide pointers to some of the open research problems.

2   The Pattern Recognition Problem

The accuracy of a machine learning technique is bounded by its pattern representation capability. The Naive Bayes Classifier, for instance, assumes that input variables are independent given the output category. In general, compared to the representation capability of the underlying language of digital computers, namely, Boolean algebra, most machine learning techniques
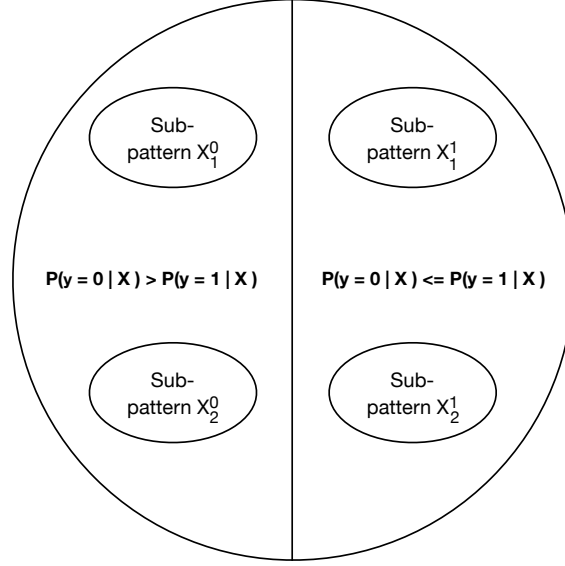
Figure 2: A partitioning of the input space according to the posterior probability of the output variable y, highlighting distinct sub-patterns in the input space, $X \in \mathcal{X}$. Sub-patterns most likely belonging to output $y = 1$ can be found on the right side, while sub-patterns most likely belonging to $y = 0$ on the left.

appear somewhat limited, neural networks being one of the exceptions. With propositional logic/Boolean algebra[1] as the starting point, we here define the pattern recognition problem to be solved by the Tsetlin Machine.

Input and Output. Consider an input vector of o propositional variables: $X = (x_1, \ldots, x_o) \in \mathcal{X}, \mathcal{X} = \{0, 1\}^o$. Together with their negated counterparts, $\bar{x}_k = \neg x_k = 1 - x_k$, the variables form a literal set $L = \{l_1, \ldots, l_{2o}\} = \{x_1, \ldots, x_o, \bar{x}_1, \ldots, \bar{x}_o\}$. From the literals, we are to produce an output vector of m propositional variables: $Y = (y^1, y^2, \ldots, y^m) \in \mathcal{Y}, \mathcal{Y} = \{0, 1\}^m$. For the sake of simple notation, we will in the remainder of the paper represent any particular output variable $y^i, i = 1, \ldots, m$, as a single output variable y, without loss of generality.

Patterns. In Tsetlin Machine learning, a pattern is expressed as a conjunctive clause in propositional logic, denoted C_j, with j being the index of the clause. A conjunctive clause is formed by ANDing a subset $L_j \subseteq L$ of the literal set:

$$C_j(X) = \bigwedge_{l_k \in L_j} l_k. \tag{4}$$

or, equivalently, as a product of the literals:

$$C_j(X) = \prod_{l_k \in L_j} l_k. \tag{5}$$

For example, the clause $C_j(X) = x_1 \wedge x_2 = x_1 x_2$ consists of the literals $L_j = \{x_1, x_2\}$ and outputs 1 iff $x_1 = x_2 = 1$.

Note that any formula in propositional logic can be reformulated as a disjunction of conjunctive clauses (referred to as disjunctive normal form). With o input variables, one can express no less than 2^{2^o} unique propositional formulas, making conjunctive clauses a powerful building block for expressing complex non-linear patterns. Arguably, they are also particularly easy for humans to comprehend [28].

---

[1]We found the Tsetlin Machine on propositional logic, which can be mapped to Boolean algebra, and vice versa.

Learning Problem and Sub-Patterns. As illustrated in Figure 2, the input space $\mathcal{X}$ can be partitioned into two parts, $\mathcal{X}^1 = \{X|P(y=0|X) \leq P(y=1|X)\}$ and $\mathcal{X}^0 = \{X|P(y=0|X) > P(y=1|X)\}$. For input vectors in $\mathcal{X}^1$, classification accuracy is maximized by assigning y the value 1. Conversely, for input vectors in $\mathcal{X}^0$, accuracy is maximized by assigning y the value 0. *Accordingly, the learning problem that we address consists of finding the partitioning of the input space that maximizes classification accuracy.*

We now come to the crucial concept of unlabelled sub-patterns. As Figure 2 exemplifies, we decompose $\mathcal{X}^0$ and $\mathcal{X}^1$ into $n/2$ sub-partitions each, respectively, $\mathcal{X}_j^0, j \in \{1, \ldots, n/2\}$, and $\mathcal{X}_j^1, j \in \{1, \ldots, n/2\}$. Each sub-partition $\mathcal{X}_j^\omega, \omega \in \{0,1\}$, is defined by a corresponding conjunctive clause, containing all the inputs that make the clause evaluate to 1, and only those inputs. Consider for instance the XOR-relation. Then $\mathcal{X}^0$ decomposes into $\mathcal{X}_1^0 = \{X|x_1 x_2 = 1\}$ and $\mathcal{X}_2^0 = \{X|\bar{x}_1 \bar{x}_2 = 1\}$, while $\mathcal{X}^1$ decomposes into $\mathcal{X}_1^1 = \{X|x_1 \bar{x}_2 = 1\}$ and $\mathcal{X}_2^1 = \{X|\bar{x}_1 x_2 = 1\}$.

During pattern learning, however, we only observe training examples (X, y) sampled from an unknown input-output distribution $P(X, y)$. Which sub-partition X was sampled from is unavailable to us. What we know is that (i) each sub-partition can be described by a conjunctive clause, and (ii) the probability of obtaining a training example from a specific sub-partition is maximally close to a known value $\frac{1}{s}, s \geq 1$, but not equal to or below. That is, we are targeting sub-patterns with a certain frequency. Formally, the pattern recognition problem that we address can be defined as follows:

Definition 1. *Let y be a random output variable taking values $\omega \in \{0,1\}$ and let $X = (x_1, \ldots, x_o)$ refer to o random input variables with domain $\mathcal{X} = \{0,1\}^o$. Given the joint input-output distribution $P(X, y)$, the input domain $\mathcal{X}$ partitions into two parts, $\mathcal{X}^1 = \{X|P(y = 0|X) \leq P(y = 1|X)\}$ and $\mathcal{X}^0 = \{X|P(y = 0|X) > P(y = 1|X)\}$. Assume that the joint probability distribution $P(X, y)$ relates y to n sub-patterns in $\mathcal{X}$, $n/2$ sub-patterns per y-value. Let each sub-pattern be defined by a corresponding conjunctive clause $Q_j^\omega(X)$, $j \in \{1, \ldots, n/2\}$, $\omega \in \{0,1\}$. That is, clause $Q_j^\omega(X)$ delineates a subset $\mathcal{X}_j^\omega$ of inputs within $\mathcal{X}$: $\mathcal{X}_j^\omega = \{X|Q_j^\omega(X) = 1\}$. Let these subsets have the following properties:*

1. *Conditioned on $\mathcal{X}^\omega$, the probability of input subset $\mathcal{X}_j^\omega$ is larger than a given constant $\frac{1}{s}, s \in [1, \infty)$: $P(\mathcal{X}_j^\omega|\mathcal{X}^\omega) > \frac{1}{s}$.*

2. *If clause $Q_j^\omega(X)$ is extended with additional literals, the resulting input subset $\mathcal{X}_j^{\omega\prime}$ becomes a subset of $\mathcal{X}_j^\omega$, $\mathcal{X}_j^{\omega\prime} \subset \mathcal{X}_j^\omega$, and occurs with probability less than $\frac{1}{s}$, conditioned on $\mathcal{X}^\omega$: $P(\mathcal{X}_j^{\omega\prime}|\mathcal{X}^\omega) < \frac{1}{s}$.*

3. *Input outside the $\mathcal{X}_j^\omega$-subsets cannot satisfy property (2) above: $P(\mathcal{X} \setminus \bigcup_j \mathcal{X}_j^\omega|\mathcal{X}^\omega) < \frac{1}{s}$.*

4. *Each input X from the subset $\mathcal{X}_j^\omega$ is a noisy predictor of the associated y-value ω, $P(y = \omega|\mathcal{X}_j^\omega) > 0.5$, implying $\mathcal{X}_j^\omega \subseteq \mathcal{X}^\omega$.*

Under these assumptions, given n and s, the pattern recognition problem consists of recovering the clauses $Q_j^\omega(X)$, $j \in \{1, \ldots n/2\}, \omega \in \{0,1\}$, only by observing input-output samples (X, y) from the unknown distribution $P(X, y)$.

Note that in practice, for real-world data, n and s may not be known. They can then be considered as hyper-parameters, optimized through a hyper-parameter search.

3 The Tsetlin Machine

The Tsetlin Machine decomposes problems into self-contained patterns that are expressed as conjunctive clauses in propositional logic. In this section, we first cover the clause-based inference structure, before we proceed to present the Tsetlin Automata teams, which are responsible for composing the clauses. The teams are coordinated by means of a game that optimizes accuracy using so-called Type I and Type II Feedback, combined with resource allocation dynamics. We conclude the section with discussing the details of the game, including pseudo code.
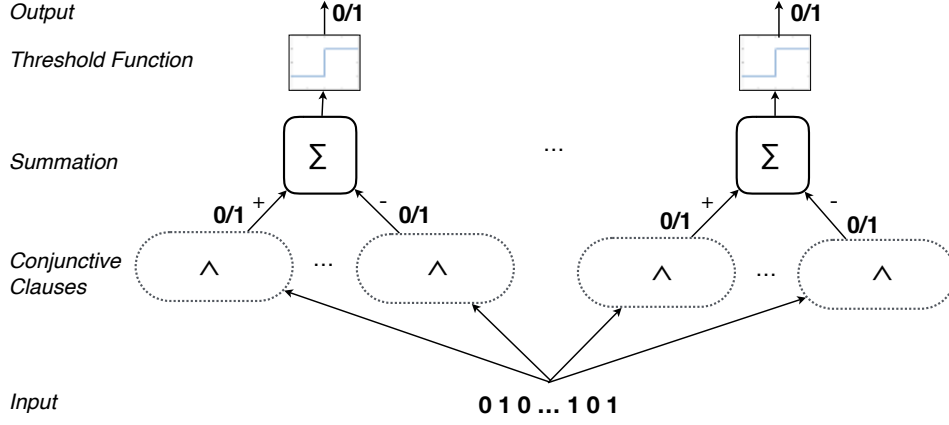
Figure 3: The Tsetlin Machine inference structure, introducing clause polarity, a summation operator collecting "votes", and a threshold function arbitrating the final output.

3.1 Tsetlin Machine Inference Structure

The Tsetlin Machine inference structure is shown in Figure 3. As seen, an input vector $X = (x_1, \ldots, x_o) \in \mathcal{X}, \mathcal{X} = \{0,1\}^o$, of o propositional variables is fed to several conjunctive clauses for evaluation.

The number of clauses employed is a user set parameter n. Half of the clauses are assigned positive polarity, denoted by upper index 1: $C_j^1, j \in \{1, \ldots, n/2\}$. The other half is assigned negative polarity, denoted by upper index 0: $C_j^0, j \in \{1, \ldots, n/2\}$. In the figure, polarity is indicated with a '+' or '-' sign, attached to each clause.

The role of the clauses is to capture sub-patterns in $\mathcal{X}$ per Eq. 4, however, with a slight adjustment. We allow clauses without literals, i.e., $L_j^\omega = \emptyset, j \in \{1, \ldots, n/2\}, \omega \in \{0,1\}$. These output 1 during learning and 0 during classification:

$$C_j^\omega(X) = \begin{cases} 1, & \textbf{if } L_j^\omega = \emptyset \textbf{ during learning,} \\ 0, & \textbf{if } L_j^\omega = \emptyset \textbf{ during classification,} \\ \bigwedge_{l_k \in L_j^\omega} l_k, & \textbf{if } L_j^\omega \neq \emptyset. \end{cases} \quad (6)$$

In other words, clauses without literals play no role during classification, and can be pruned from the inference structure. During learning, however, outputting 1 stimulates further updating of the clause, as explored in Section 3.3.

The clause evaluations, in turn, are combined into a final output decision through summation and unit step thresholding, $u(v) = 1$ **if** $v \geq 0$ **else** 0:

$$\hat{y} = u\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X) \right). \quad (7)$$

That is, output is decided based on a majority vote, with the positive polarity clauses voting for $y = 1$ and the negative ones for $y = 0$. The purpose is to reach a balanced output decision, weighting positive against negative evidence, found in the input. The classifier $\hat{y} = u(x_1\bar{x}_2 + \bar{x}_1 x_2 - x_1 x_2 - \bar{x}_1\bar{x}_2)$, for instance, captures the XOR-relation.

3.2 The Tsetlin Automata Team for Composing Clauses

Recall that a Tsetlin Automaton can be represented as a quintuple $\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, F(\cdot, \cdot), G(\cdot)\}$. Each clause $C_j^\omega, \omega \in \{0,1\}, j \in \{1, \ldots, n/2\}$, of a Tsetlin Machine is composed by a team of such Tsetlin Automata. The team consists of $2o$ automata, one assigned per literal $l_k \in L$. Let $\Phi_{jk}^\omega \in \{\phi_1, \phi_2 \ldots, \phi_{2N}\}$ refer to the state of the Tsetlin Automaton in charge of literal l_k for
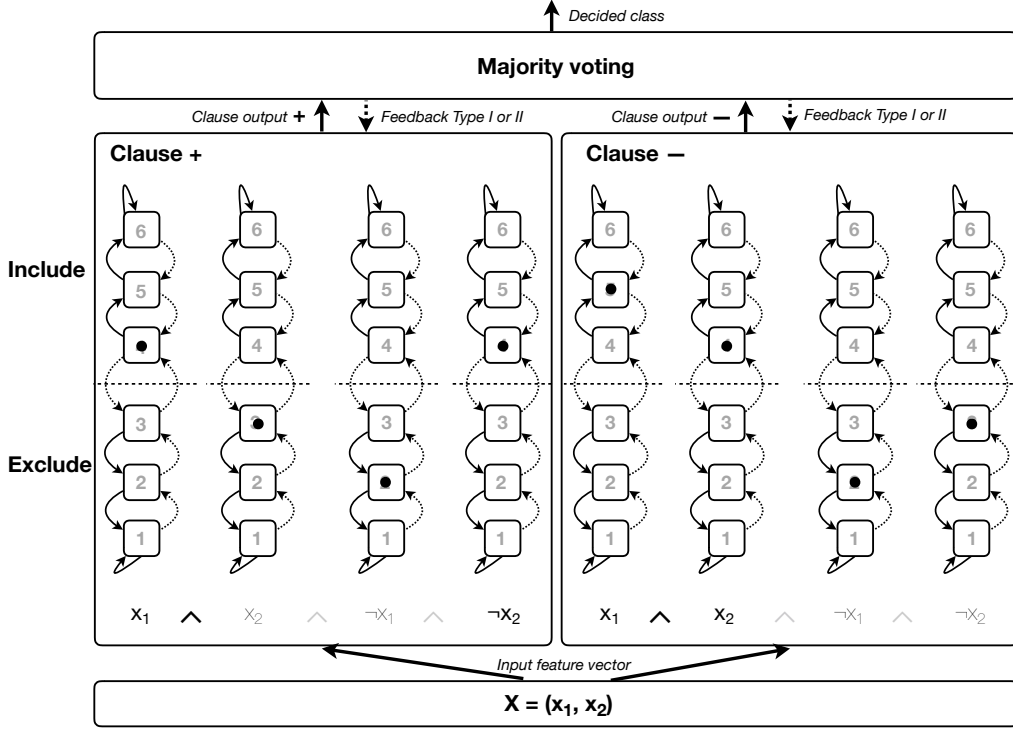
8

Figure 4: Two Tsetlin Automata teams, each producing a conjunctive clause. The overall output is based on majority voting.

clause $C_j^\omega, \omega \in \{0, 1\}, j \in \{1, \ldots, n/2\}$. It is this state that decides whether the literal is excluded (action α_1) or included (action α_2) in the clause.

Figure 4 depicts two clauses, one with positive polarity (C^1) and one with negative polarity (C^0). They are both assigned four Tsetlin Automata, each with six states, controlling the inclusion of literals $L = \{x_1, x_2, \neg x_1, \neg x_2\}$. The leftmost Tsetlin Automaton in the figure has for example decided to include literal x_1 because it is in state ϕ_4. The second Tsetlin Automaton, being in state ϕ_3, has decided to exclude literal x_2. Given the decisions of the four Tsetlin Automata of C^1, the clause has the form $x_1 \bar{x}_2$. Similarly, clause C^0 obtains the form $x_1 x_2$.

3.3 The Tsetlin Machine Game for Learning Conjunctive Clauses

We here introduce the game theoretic learning mechanism that guides the Tsetlin Automata towards solving the pattern recognition problem from Definition 1. The game is designed to deal with the problem of vanishing signal-to-noise ratio for learning automata games (cf. Section 1.3).

3.3.1 Tsetlin Automata Games

A game of Tsetlin Automata involves multiple automata and is played over several rounds [3]. In each round of the game, the participating Tsetlin Automata independently decide upon their next action from $\underline{\alpha}$. Thus, if two actions are available to each automaton, they jointly select among 2^W unique action configurations, with W being the number of automata. After the Tsetlin Automata have made their decisions, the round ends with the automata being individually penalized/rewarded based on the jointly selected action configuration. The next round of the game then starts.

To fully specify the game, we need to assign a reward probability per Tsetlin Automaton for each unique configuration of actions. We refer to these reward probabilities as the *payoff matrix* of the game. With W two-action automata, we accordingly need $W2^W$ reward probabilities for the payoff matrix.

3.3.2 Design of the Payoff Matrix

The potential complexity of the Tsetlin Machine game is immense, because the decisions of every single Tsetlin Automaton jointly decide the behaviour of the Tsetlin Machine as a whole. Indeed, under the right conditions, a single Tsetlin Automaton has the power to completely disrupt a clause by introducing a contradiction. The payoffs of the game must therefore be carefully designed so that the Tsetlin Automata are guided towards building the clauses that solve the pattern recognition problem at hand. To complicate further, an explicit enumeration of the payoffs is impractical due to the potentially tremendous size of the payoff matrix.

To address the above challenges, we decompose the payoff matrix based on the notion of true positive, false negative, and false positive clause output, treating clauses as individual classifiers. True negative clause output is ignored to increase the freedom of the automata. By progressively suppressing false negative and false positive clause output, and reinforcing true positive clause output, we intend to guide the Tsetlin Automata towards maximizing pattern recognition accuracy. This guiding is based on what we will refer to as Type I and Type II Feedback. Based on the principles exposed in this section, we will define the resulting game formally in Section 4.

3.3.3 Type I Feedback – Combating False Negative Output

In each round of the game, the Tsetlin Machine receives an input-output sample (X, y). Type I Feedback is then given to the Tsetlin Automata of clauses C_j^ω when $y = \omega$. Consider a particular Tsetlin Automaton with state Φ_{jk}^ω, controlling inclusion of literal l_k in clause C_j^ω, i.e., the jth clause of polarity ω. Feedback is then decided by three factors:

1. The decisions of the Tsetlin Automata team for clause $C_j^\omega(X)$ as a whole, manifested by the output of the clause.

2. The truth value of the literal l_k assigned to the automaton.

3. The action $\alpha_z = G(\Phi_{jk}^\omega), z \in \{1, 2\}$, decided by the automaton.

Table 2 contains the probabilities that we use to generate Type I Feedback, looking up the above three factors. For instance, assume that:

1. Clause $C_j^\omega(X)$ evaluates to 1,

2. Literal l_k is 1, and

3. Automaton state Φ_{jk}^ω produces the *Include* action (α_2).

By examining the corresponding cell in Table 2, we observe that the probability of receiving a reward $P(\text{Reward})$ is $\frac{s-1}{s}$. Further, the probability of inaction $P(\text{Inaction})$ is $\frac{1}{s}$. Finally, the probability of receiving a penalty $P(\text{Penalty})$ is zero.

Note that the Inaction feedback is a novel extension to the Tsetlin Automaton, which traditionally receives either a Reward or a Penalty. When receiving the Inaction feedback, the Tsetlin Automaton is simply left unchanged.

Brief Analysis of Type I Feedback. Notice how the Type I Feedback probabilities force production of clauses that contain many literals, up to a certain point. We achieve this by reinforcing *Include* actions more strongly than *Exclude* actions, using reward- and penalty probabilities favouring *Include*. As seen, the hyper-parameter $s, s \geq 1$, controls how strongly we favour *Include*. In effect, s decides how "fine-grained" patterns the clauses are going to capture. The larger the value of s, the more the Tsetlin Automata are stimulated to include literals in their clauses. The countering force is non-matching input, i.e., input that makes the clause evaluate to 0. For such input, the *Exclude* action is rewarded and the *Include* action is penalized. Clearly, the probability of encountering non-matching input grows as s increases (more literals are included). When the above two counter-acting forces are in balance, we have

a Nash equilibrium as discussed further in Section 4. These dynamics are a critical part of the Tsetlin Machine, allowing learning of any sub-pattern, no matter how infrequent, as decided by s.

Boosting of True Positive Feedback (Column 1 in Table 2). The feedback probabilities in Table 2 have been selected based on mathematical derivations (see Section 4). For certain real-life data sets, however, it turns out that further boosting of *Include* actions can be beneficial. That is, pattern recognition accuracy can be enhanced by boosting rewarding of these actions when they produce true positive outcomes. Penalizing of *Exclude* actions is then adjusted accordingly. In all brevity, we boost rewarding in this manner by replacing $\frac{s-1}{s}$ with 1.0 and $\frac{1}{s}$ with 0.0 in Column 1 of Table 2.

3.3.4 Type II Feedback – Combating False Positive Output

Table 3 covers Type II Feedback, that is, feedback that combats false positive output. Type II Feedback is given to the Tsetlin Automata of clauses C_j^ω when $y \neq \omega$. The purpose of this feedback is to introduce candidate literals that increase discrimination power. That is, a clause C_j^ω is supposed to output 0 when $y \neq \omega$, to discriminate between the outputs $\omega \in \{0,1\}$. However, if it erroneously evaluates to 1, we correct this by identifying the Tsetlin Automata that have excluded a zero-valued literal from the clause. It is sufficient to merely include one of these literals to make the offending clause evaluate to 0 instead. This is because the clauses are conjunctive.

Brief Analysis of Type II Feedback. Observe that Type II Feedback only penalizes the *exclusion* of literals of value 0 when a clause outputs 1 (false positive output). When a clause outputs 0 (the two leftmost columns in the table), only Inaction-feedback is given. This is to avoid local minima by leaving it to Type I feedback to reinforce *Include* actions. Accordingly, together, Type I Feedback and Type II Feedback interact to minimize the expected output error, moving towards a global optimum as explored in Section 4.

Truth Value of Clause C_j^ω		1		0	
Truth Value of Literal l_k		1	0	1	0
Include Literal ($\alpha_2 \to l_k \in L_j^\omega$)	$P(\text{Reward})$	$\frac{s-1}{s}$	NA	0	0
	$P(\text{Inaction})$	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	$P(\text{Penalty})$	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
Exclude Literal ($\alpha_1 \to l_k \notin L_j^\omega$)	$P(\text{Reward})$	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	$P(\text{Inaction})$	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	$P(\text{Penalty})$	$\frac{s-1}{s}$	0	0	0

Table 2: Type I Feedback — Feedback from the perspective of a single Tsetlin Automaton deciding to either *Include* or *Exclude* a given literal l_k in the clause C_j^ω. Type I Feedback is triggered to increase the number of clauses that correctly evaluates to 1 for a given input X.

Truth Value of Clause C_j^ω		1		0	
Truth Value of Literal l_k		1	0	1	0
Include Literal ($\alpha_2 \to l_k \in L_j^\omega$)	$P(\text{Reward})$	0	NA	0	0
	$P(\text{Inaction})$	1.0	NA	1.0	1.0
	$P(\text{Penalty})$	0	NA	0	0
Exclude Literal ($\alpha_1 \to l_k \notin L_j^\omega$)	$P(\text{Reward})$	0	0	0	0
	$P(\text{Inaction})$	1.0	0	1.0	1.0
	$P(\text{Penalty})$	0	1.0	0	0

Table 3: Type II Feedback — Feedback from the perspective of a single Tsetlin Automaton deciding to either *Include* or *Exclude* a given literal l_k in the clause C_j^ω. Type II Feedback is triggered to increase the discrimination power of the clauses.

3.3.5 Allocating Clauses to Sub-Patterns

For the Tsetlin Machine output to be correct it is sufficient that the summation part of Eqn. 7, $\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X)$, has the correct sign. That is, it is sufficient that a single clause, with the appropriate polarity, outputs 1. In the case of noisy data, however, it can be beneficial to introduce a margin T, referred to as the target of the summation. The intent is to make the available clauses distribute themselves across the sub-patterns in the data, so that T clauses of the correct polarity evaluate to 1 for any particular input X. Additionally, the summation target T opens up for interplay between the clauses, including rectification of special cases.

The resource allocation mechanism that we introduce here is inspired by a finite state automaton-based solution to the fractional knapsack problem in unknown and stochastic environments [29]. The purpose of the mechanism is to ensure that the Tsetlin Machine only spends a few of the available clauses to represent each specific sub-pattern. This is achieved by randomly selecting clauses to receive feedback, reducing intensity when approaching the summation target T.

Generating Type I Feedback. For Type I Feedback, we randomly select clauses $C_j^\omega, \omega = y$. Among these, the probability of receiving Type I Feedback is

$$\frac{T - \mathbf{clip}\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X), -T, T\right)}{2T} \tag{8}$$

when $y = 1$, and

$$\frac{T + \mathbf{clip}\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X), -T, T\right)}{2T} \tag{9}$$

when $y = 0$. Here, the clip operation restricts the sum to lie between $-T$ and T.

Generating Type II Feedback. For Type II Feedback, we randomly select clauses $C_j^\omega, \omega \neq y$. The probability of receiving Type II Feedback is:

$$\frac{T + \mathbf{clip}\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X), -T, T\right)}{2T} \tag{10}$$

when $y = 1$, and

$$\frac{T - \mathbf{clip}\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X), -T, T\right)}{2T} \tag{11}$$

when $y = 0$.

Notice how the feedback vanishes as the number of triggering clauses correctly approaches $T/-T$. This is pertinent for effective use of the available pattern representation capacity. To exemplify, assume that the correct output is $y = 1$ for an input X. If the clause output sum accumulates to a total of T or more, neither rewards nor penalties are provided to the involved Tsetlin Automata. This leaves the Tsetlin Automata free to learn other sub-patterns.

3.3.6 The Tsetlin Machine Algorithm

The step-by-step procedure for learning conjunctive clauses can be found in Algorithm 1. The algorithm takes a set of training examples $(X, y) \in \mathcal{S}$ as input. It then produces a collection of conjunctive clauses for predicting the output y. We will now take a closer look at the algorithm, line-by-line.

Lines 2-3. From the perspective of game theory, we form one team of Tsetlin Automata per clause, denoted $\mathcal{A}_j^0$, $\mathcal{A}_j^1$, $j \in \{1, \ldots, n/2\}$. A team consists of $2o$ automata, each assigned a user specified number of states, $\Phi_{jk}^\omega \in \{\phi_1, \phi_2, \ldots, \phi_{2N}\}$, $j \in \{1, \ldots, n/2\}$, $k \in \{1, \ldots, 2o\}$, $\omega \in \{0, 1\}$. The initial state is then set to $\Phi_{jk}^\omega \leftarrow \phi_N$ (the *Exclude* action).

Line 5. The learning process is driven by a set of training examples $\mathcal{S}$, sampled from the input-output distribution $P(X, y)$. Each training example (X, y) is fed to the Tsetlin Machine, one at a time, facilitating online learning.

Algorithm 1 Tsetlin Machine

Input Training data $(X, y) \in \mathcal{S} \sim P(X, y)$, Number of clauses n, Number of inputs o, Precision s, Target T

Output Trained conjunctive clauses $C_j^1, C_j^0, j = 1, \ldots, n/2$

1: **function** TRAINTSETLINMACHINE($\mathcal{S}, n, o, s, T$)
2:     $\mathcal{A}_1^1, \ldots, \mathcal{A}_{n/2}^1 \leftarrow$ CreateTATeams($n/2, 2o, \phi_N$)   ▷ Create $n/2$ teams of $2o$ Tsetlin Automata (TA), one team $\mathcal{A}_j^1$ per clause C_j^1. Each TA is initialized to state ϕ_N.
3:     $\mathcal{A}_1^0, \ldots, \mathcal{A}_{n/2}^0 \leftarrow$ CreateTATeams($n/2, 2o, \phi_N$)   ▷ Create another $n/2$ teams of $2o$ TA, one team $\mathcal{A}_j^0$ per clause C_j^0. Each TA is initialized to state ϕ_N.
4:     **repeat**
5:         $X, y \leftarrow$ GetTrainingExample($\mathcal{S}$)   ▷ Mini-batches, random selection, etc.
6:         $C_1^1, \ldots, C_{n/2}^1 \leftarrow$ ObtainClauses($\mathcal{A}_1^1, \ldots, \mathcal{A}_{n/2}^1$)   ▷ Each TA team $\mathcal{A}_j^1$ produces a clause C_j^1.
7:         $C_1^0, \ldots, C_{n/2}^0 \leftarrow$ ObtainClauses($\mathcal{A}_1^0, \ldots, \mathcal{A}_{n/2}^0$)   ▷ Each TA team $\mathcal{A}_j^0$ produces a clause C_j^0.
8:         $v \leftarrow \sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X)$ ▷ Calculate sum of clause outputs.
9:         **for** $j \leftarrow 1, \ldots, n/2$ **do**   ▷ Give feedback to the TA teams.
10:             **if** $y = 1$ **then**
11:                 **if** Random() $\leq \frac{T - \mathbf{clip}(v, -T, T)}{2T}$ **then**
12:                     GenerateTypeIFeedback($X, C_j^1(X), \mathcal{A}_j^1, o$)   ▷ Give Type I Feedback to TA of clause C_j^1.
13:                 **end if**
14:                 **if** Random() $\leq \frac{T - \mathbf{clip}(v, -T, T)}{2T}$ **then**
15:                     GenerateTypeIIFeedback($X, C_j^0(X), \mathcal{A}_j^0, o$)   ▷ Give Type II Feedback to TA of clause C_j^0.
16:                 **end if**
17:             **else if** $y = 0$ **then**
18:                 **if** Random() $\leq \frac{T + \mathbf{clip}(v, -T, T)}{2T}$ **then**
19:                     GenerateTypeIIFeedback($X, C_j^1(X), \mathcal{A}_j^1, o$)   ▷ Give Type II Feedback to TA of clause C_j^1.
20:                 **end if**
21:                 **if** Random() $\leq \frac{T + \mathbf{clip}(v, -T, T)}{2T}$ **then**
22:                     GenerateTypeIFeedback($X, C_j^0(X), \mathcal{A}_j^0, o$)   ▷ Give Type I Feedback to TA of clause C_j^0.
23:                 **end if**
24:             **end if**
25:         **end for**
26:     **until** StopCriteria($\mathcal{S}, C_1^0, \ldots, C_{n/2}^0, C_1^1, \ldots, C_{n/2}^1$)
27:     **return** PruneAllExcludeClauses($C_1^0, \ldots, C_{n/2}^0, C_1^1, \ldots, C_{n/2}^1$)   ▷ Return completely trained conjunctive clauses after pruning clauses where all literals have been excluded.
28: **end function**

Algorithm 2 Type I Feedback

Input Input vector X of size o, Clause output c_j^ω, Tsetlin Automata team $\mathcal{A}_j^\omega$

1: **procedure** GENERATETYPEIFEEDBACK($X, c_j^\omega, \mathcal{A}_j^\omega, o$)
2:     **for** $k \leftarrow 1, \ldots, 2o$ **do**   ▷ Reward/Penalize all Tsetlin Automata in $\mathcal{A}_j^\omega$.
3:          $l_k \leftarrow$ ObtainLiteral(X, k)   ▷ Get literal l_k from input X.
4:          $\alpha_u \leftarrow G(\Phi_{jk}^\omega)$ ▷ Obtain corresponding automaton action according to Eqn. 1.
5:          $\beta_v \leftarrow$ SampleTypeIFeedback($\alpha_u, l_k, c_j^\omega$)   ▷ Sample Type I Feedback from Table 2.
6:          $\Phi_{jk}^\omega \leftarrow F(\Phi_{jk}^\omega, \beta_v)$ ▷ Update state of Tsetlin Automaton according to Eqn. 2.
7:     **end for**
8: **end procedure**

Algorithm 3 Type II Feedback

Input Input vector X of size o, Clause output c_j^ω, Tsetlin Automata team $\mathcal{A}_j^\omega$

1: **procedure** GENERATETYPEIIFEEDBACK($X, c_j^\omega, \mathcal{A}_j^\omega, o$)
2:     **for** $k \leftarrow 1, \ldots, 2o$ **do**   ▷ Penalize all Tsetlin Automata in $\mathcal{A}_j^\omega$.
3:          $l_k \leftarrow$ ObtainLiteral(X, k)   ▷ Get literal l_k from input X.
4:          $\alpha_u \leftarrow G(\Phi_{jk}^\omega)$ ▷ Obtain corresponding automaton action according to Eqn. 1.
5:          $\beta_v \leftarrow$ SampleTypeIIFeedback($\alpha_u, l_k, c_j^\omega$)   ▷ Sample Type II Feedback from Table 3.
6:          $\Phi_{jk}^\omega \leftarrow F(\Phi_{jk}^\omega, \beta_v)$ ▷ Update state of Tsetlin Automaton according to Eqn. 2.
7:     **end for**
8: **end procedure**

Line 6-7. In each iteration, the Tsetlin Automata decide whether to *Include* or *Exclude* their assigned literals in their respective clauses. The result is a new set of clauses, $C_1^0, \ldots, C_{n/2}^0$, $C_1^1, \ldots, C_{n/2}^1$, for predicting y.

Line 8. Based on the new clauses, the current example X is evaluated, producing a clause sum v according to Eqn. 7. The unit step function u is not used here because examples are not classified during training.

Lines 9-25. The next step is to randomly determine the clauses whose Tsetlin Automata are to receive Type I Feedback (Eqn. 8-9) and Type II Feedback (Eqn. 10-11), based on the clipped difference between v and T.

After Type I or Type II Feedback have been triggered for a clause, the Tsetlin Automata of the clause are rewarded/penalized according to Algorithm 2 and Algorithm 3. In all brevity, rewarding/penalizing is directly based on Table 2 and Table 3.

Line 26. The above steps are iterated until a stopping criteria is fulfilled (for instance a certain number of iterations over the dataset). The current clauses are then returned as the output of the learning process, after the clauses without literals have been removed.

3.4 Implementation Using Bitwise Operators

Small memory footprint and speed of operation can be crucial in complex and large scale pattern recognition. Being based on propositional formula, the Tsetlin Machine architecture can naturally be represented with bits and manipulated upon using bitwise operators. However, it is not straightforward how to represent and update the Tsetlin Automata themselves. First of all, the state index of each Tsetlin Automaton is an integer value. Further, the action of an automaton is decided upon using a smaller-than operator, while feedback is processed by means of increment and decrement operations.

One approach to bitwise operation is to jointly represent the state indexes of all of the Tsetlin Automata of a clause C_j^ω with multiple sequences of bits. Sequence 1 then contains the first bit of each state index, sequence 2 contains the second bit, and so on, as exemplified in Figure 5 for 24 Tsetlin Automata. The benefit of this representation is that the action of each Tsetlin Automaton is readily available from the most significant bit (sequence 8 in the figure).

Thus, the output of the clause can be obtained from the input based on fast bitwise operators (NOT, AND, and CMP - comparison).

In the figure, a setup for the Noisy XOR dataset from Section 5 is used for illustration purposes. First, the 12-bit input is extended to 24 bits by concatenating the original input with the original input inverted. The resulting 24 bits are in turn connected with 24 Tsetlin Automata. The first 12 control the non-negated input, while the second 12 control the negated input.

Employing the latter bit-based representation reduces memory usage four times, compared to using a full 32-bit integer to represent the state of each and every Tsetlin Automaton. More importantly, it is possible to increment/decrement the states of all of the automata in parallel with bitwise operations through customized increment/decrement procedures, significantly increasing learning speed. As an example, for the MNIST dataset (cf. Section 5), the overall memory usage is approximately ten times smaller, learning speed 3.5 times faster, and classification speed 8 times faster with the bit-based representation.

When deployed after training, only the state bit sequence containing the most significant bit is required. The other bit sequences can be discarded because these bits are only used to keep track of the learning. This provides a further reduction in memory usage.
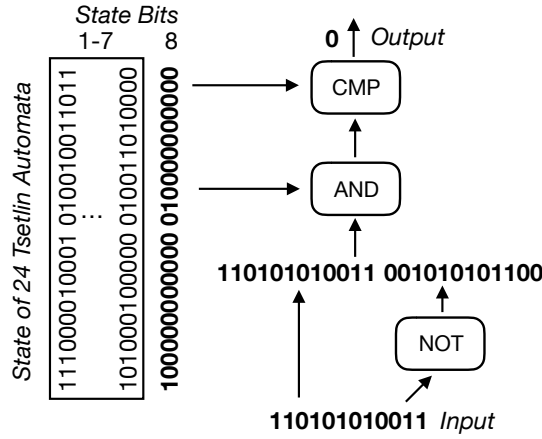

Figure 5: Bit-based representation of a clause for the Noisy XOR dataset (see Section 5). The bit-based representation of the Tsetlin Automata states allows the actions of all of the automata to be obtained directly from the most significant bit (bit 8 in the figure).

4 Theoretical Analysis

In this section, we analyse the Tsetlin Machine game formally. We base the analysis on partitioning the input space $\mathcal{X}$ into subsets that isolate the impact a single Tsetlin Automaton has on classification, mediated through its clause. We then study each of these subsets for the two output scenarios $y = 0$ and $y = 1$. From this investigation, we identify the Nash equilibria of Tsetlin Machine learning and compare the Nash equilibria clauses with the optimal sub-patterns from Definition 1.

4.1 The Payoff Matrix of the Tsetlin Machine Game

Based on the learning dynamics described algorithmically in Section 3, we now introduce a more succinct description of Tsetlin Machine learning by formally defining the payoff matrix of the Tsetlin Machine game.

Definition 2. *Consider a pattern recognition problem per Definition 1 and a Tsetlin Machine with n clauses $C_j^\omega(X), j \in \{1, \ldots, n/2\}, \omega \in \{0, 1\}$. The resulting game consists of*

$2no$ *Tsetlin Automata (players), one automaton per literal* $l_k \in L = \{x_1, \ldots, x_o, \bar{x}_1, \ldots, \bar{x}_o\}$, *per clause* $C_j^\omega(X)$. *Let* A_{jk}^ω *be a Tsetlin Automaton with actions* $a_{jk}^\omega \in \{\alpha_1, \alpha_2\}$, *controlling the inclusion of literal* l_k *in clause* C_j^ω (α_1 *represents Exclude and* α_2 *represents Include). Each cell in the payoff matrix of the game then refers to a unique action configuration* $\overline{\alpha} = (a_{1,1}^0, \ldots, a_{n/2,2o}^0, a_{1,1}^1 \ldots, a_{n/2,2o}^1) \in \{\alpha_1, \alpha_2\}^{2no}$. *Accordingly, the payoff matrix consists of* 2^{2no} *cells. For every cell, we define the payoff of each* a_{jk}^ω*-action in* $\overline{\alpha}$ *using the four stochastic variables* X, y, U_j^ω, *and* R_{jk}^ω:

- *Each game round, the distribution* $P(X, y)$ *randomly produces a training example* (X, y).

- *Let the binary stochastic variable* U_j^ω *refer to whether clause* C_j^ω *is updated* ($U_j^\omega = 1$) *or ignored* ($U_j^\omega = 0$) *in the given game round. The distribution of* U_j^ω *is decided as follows. First, the clipped clause sum* $v^c = \mathbf{clip}\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X), -T, T\right)$ *is determined. This clipped clause sum is calculated from the clauses produced by the joint action configuration* $\overline{\alpha}$ *of the payoff matrix cell, evaluated upon* X. *Clause* C_j^ω *is then updated with probability* $P(U_j^\omega = 1) = \frac{T-v^c}{2T}$ *if* $y = 1$ *and with probability* $P(U_j^\omega = 1) = \frac{T+v^c}{2T}$ *if* $y = 0$.

- *Finally, we decide the payoff* $R_{jk}^\omega \in \{-1, 0, +1\}$ *(Penalty, Inaction, Reward) for each automaton* A_{jk}^ω. *If* $U_j^\omega = 0$ *the payoff is* 0. *Otherwise,* R_{jk}^ω *is randomly set based on Table 2 if* $y = \omega$ *(Type I Feedback) and Table 3 if* $y \neq \omega$ *(Type II Feedback). We obtain the distribution of* R_{jk}^ω *from the designated table by looking up: (1) the action selected* (α_1 *or* α_2), *(2) the value of literal* l_k *from* X, *and (3) the value of the clause* $C_j^\omega(X)$.

Accordingly, the payoff of each action a_{jk}^ω *in* $\overline{\alpha}$ *becomes* $U_j^\omega \cdot R_{jk}^\omega$.

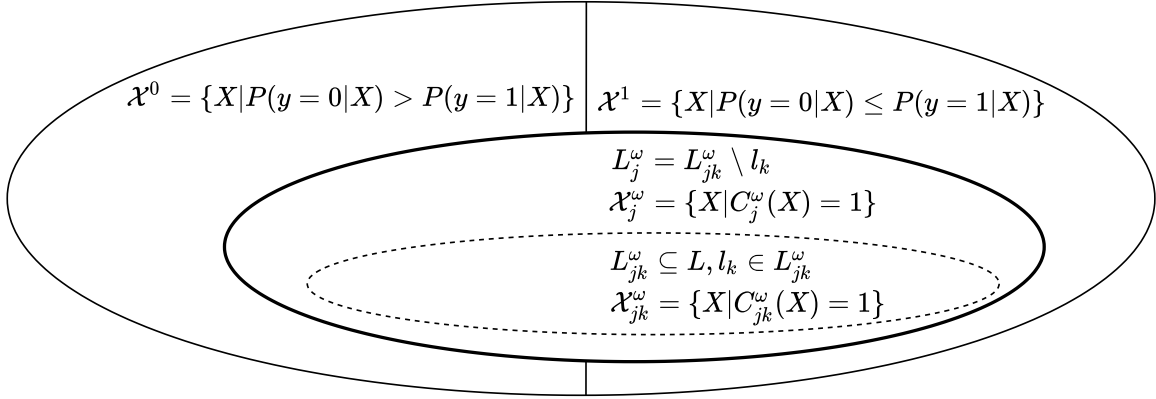In the following, we analyze the dynamics of the game using the above payoff matrix.

Figure 6: The subsets $\mathcal{X}^0$, $\mathcal{X}^1$, $\mathcal{X}_j^\omega$, and $\mathcal{X}_{jk}^\omega$, pertinent for the theoretical analysis.

4.2 Pertinent Input Space Subsets

The input space subsets ($\mathcal{X}$-subsets) of interest are listed in Table 4 and Table 5. Observe that all of the $\mathcal{X}$-subsets in the tables are derived from the following four subsets: $\mathcal{X}^0$, $\mathcal{X}^1$, $\mathcal{X}_j^1$, and $\mathcal{X}_{jk}^1$, illustrated in Figure 6.

The subset $\mathcal{X}^1 = \{X|P(y=0|X) \leq P(y=1|X)\}$ is the subset of the input space where it is optimal to output $y = 1$. Conversely, the subset $\mathcal{X}^0 = \{X|P(y=0|X) > P(y=1|X)\}$ is the compliment of $\mathcal{X}$, where it is optimal to output $y = 0$. The subset $\mathcal{X}_{jk}^1 = \{X|C_{jk}^\omega(X) = 1\}$, $L_{jk}^\omega \subseteq L, l_k \in L_{jk}$, is the part of the input space in which the clause $C_{jk}^\omega(X)$ with literals $L_{jk}^\omega, l_k \in L_{jk}^\omega$, evaluates to 1. Finally, the subset $\mathcal{X}_j^1 = \{X|C_j^\omega(X) = 1\}, L_j^\omega = L_{jk}^\omega \setminus l_k$, is a superset of $\mathcal{X}_{jk}^1$, where the literal l_k has been removed from the conjunction.

Recall that it is a single Tsetlin Automaton A_{jk}^ω that decides whether to *Include* or *Exclude* literal l_k above, which means that it in effect decides between clause $C_{jk}^\omega(X)$ (literal included) and clause $C_j^\omega(X)$ (literal excluded).

Further note that we in this section ignore polarity to simplify notation. That is, we analyze how the clauses with positive polarity in the Tsetlin Machine game capture the sub-patterns $Q_h^1, h \in \{1, \ldots, n/2\}$, from Definition 1. The analysis for negative polarity clauses follows the same structure.

4.3 Expected Payoff for the Actions of a Single Tsetlin Automaton

As defined in the payoff matrix of the Tsetlin Machine game, the stochasticity in Tsetlin Machine learning comes from: (i) the training samples (X, y), which are drawn randomly according to an unknown distribution $P(X, y)$; (ii) the random selection of clauses for updating; and (iii) the random generation of the rewards and penalties from Type I feedback (Table 2). *We first focus on what happens with a specific Tsetlin Automaton each time its clause activates. Thus, for the moment, we ignore stochasticity stemming from random clause selection, studying a single clause.*

We have collected the information needed to calculate the expected payoff for the *Exclude* action in Table 4 and for the *Include* action in Table 5. The first column of the table enumerates the possible y-values. The second column specifies a partitioning of the input space $\mathcal{X}$ for each y-value. The third column contains the expected payoff of the action covered by the table, for the given $\mathcal{X}$-subset and output y. The expected payoff has been obtained from Table 2 and Table 3. The fourth column states the probability of obtaining a training example (X, y)

y	$\mathcal{X}$-Subset	R	Probability	Noise $1-\gamma$
0	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^1$	0	$P(y=0\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$	$(1-\gamma)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$
0	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^1$	-1	$P(y=0\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$	$(1-\gamma)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$
0	$\mathcal{X}^1 \setminus \mathcal{X}_j^1$	0	$P(y=0\|\mathcal{X}^1 \setminus \mathcal{X}_j^1)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$	$(1-\gamma)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$
0	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^0$	0	$P(y=0\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$	$\gamma P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$
0	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^0$	-1	$P(y=0\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$	$\gamma P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$
0	$\mathcal{X}^0 \setminus \mathcal{X}_j^1$	0	$P(y=0\|\mathcal{X}^0 \setminus \mathcal{X}_j^1)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$	$\gamma P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$
1	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^1$	$-\frac{s-1}{s}$	$P(y=1\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$	$\gamma P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$
1	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^1$	$\frac{1}{s}$	$P(y=1\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$	$\gamma P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$
1	$\mathcal{X}^1 \setminus \mathcal{X}_j^1$	$\frac{1}{s}$	$P(y=1\|\mathcal{X}^1 \setminus \mathcal{X}_j^1)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$	$\gamma P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$
1	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^0$	$-\frac{s-1}{s}$	$P(y=1\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$	$(1-\gamma)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$
1	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^0$	$\frac{1}{s}$	$P(y=1\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$	$(1-\gamma)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$
1	$\mathcal{X}^0 \setminus \mathcal{X}_j^1$	$\frac{1}{s}$	$P(y=1\|\mathcal{X}^0 \setminus \mathcal{X}_j^1)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$	$(1-\gamma)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$

Table 4: Expected payoff for the *Exclude* action.

y	$\mathcal{X}$-Subset	R	Probability	Noise $1-\gamma$
0	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^1$	0	$P(y=0\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$	$(1-\gamma)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$
0	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^1$	0	$P(y=0\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$	$(1-\gamma)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$
0	$\mathcal{X}^1 \setminus \mathcal{X}_j^1$	0	$P(y=0\|\mathcal{X}^1 \setminus \mathcal{X}_j^1)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$	$(1-\gamma)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$
0	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^0$	0	$P(y=0\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$	$\gamma P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$
0	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^0$	0	$P(y=0\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$	$\gamma P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$
0	$\mathcal{X}^0 \setminus \mathcal{X}_j^1$	0	$P(y=0\|\mathcal{X}^0 \setminus \mathcal{X}_j^1)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$	$\gamma P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$
1	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^1$	$\frac{s-1}{s}$	$P(y=1\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$	$\gamma P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$
1	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^1$	$-\frac{1}{s}$	$P(y=1\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$	$\gamma P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1)$
1	$\mathcal{X}^1 \setminus \mathcal{X}_j^1$	$-\frac{1}{s}$	$P(y=1\|\mathcal{X}^1 \setminus \mathcal{X}_j^1)P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$	$\gamma P(\mathcal{X}^1 \setminus \mathcal{X}_j^1)$
1	$\mathcal{X}_{jk}^1 \cap \mathcal{X}^0$	$\frac{s-1}{s}$	$P(y=1\|\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$	$(1-\gamma)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0)$
1	$(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap \mathcal{X}^0$	$-\frac{1}{s}$	$P(y=1\|[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$	$(1-\gamma)P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)$
1	$\mathcal{X}^0 \setminus \mathcal{X}_j^1$	$-\frac{1}{s}$	$P(y=1\|\mathcal{X}^0 \setminus \mathcal{X}_j^1)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$	$(1-\gamma)P(\mathcal{X}^0 \setminus \mathcal{X}_j^1)$

Table 5: Expected payoff for the *Include* action.

for the corresponding $\mathcal{X}$-subset and output y. The fifth column specifies these probabilities assuming a static output noise probability $P(y = 0|\mathcal{X}^1) = P(y = 1|\mathcal{X}^0) = 1 - \gamma, \gamma > 0.5$. If for instance $\gamma = 0.75$, the output is erroneously $y = 0$ for $X \in \mathcal{X}^1$ and $y = 1$ for $X \in \mathcal{X}^0$, on average, 25% of the time. The assumption of static noise is not a limitation of the Tsetlin Machine, however, the assumption simplifies the theoretical analysis that follows.

From Table 4 and Table 5, we can calculate the expected payoff of each action by summing up the rewards (positive polarity) and penalties (negative polarity) in the **R**-column, each multiplied with the probability of the $\mathcal{X}$-subset and y-value of its row. The addend from the first row in Table 4 is for instance $0 \cdot P(y = 0|\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$, which becomes zero because the reward is zero.

Lemma 1. *Consider a literal l_k and clause C_j^ω. The expected payoff of excluding l_k from C_j^ω in the Tsetlin Machine game (Definition 2) is:*

$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) + \frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1) - \frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad -$$
$$\frac{s-1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) - (1 - \gamma) \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) - \gamma \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0). \quad (12)$$

Proof. We prove Lemma 1 based on Table 4. The expected payoff of the *Exclude* action can be calculated as follows, assuming static noise probability $1 - \gamma$:

$$\frac{1}{s} \cdot \gamma \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) \quad +$$
$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_j^1) \quad +$$
$$\frac{1}{s} \cdot (1 - \gamma) \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) \quad +$$
$$\frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_j^1) \quad -$$
$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad -$$
$$\frac{s-1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) \quad -$$
$$(1 - \gamma) \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) \quad -$$
$$\gamma \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0). \quad (13)$$

Since $(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cap (\mathcal{X}^1 \setminus \mathcal{X}_j^1) = \emptyset$ and $(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \cup (\mathcal{X}^1 \setminus \mathcal{X}_j^1) = \mathcal{X}^1 \setminus \mathcal{X}_{jk}^1$, we can simplify to:

$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) \quad +$$
$$\frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1) \quad -$$
$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad -$$
$$\frac{s-1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) \quad -$$
$$(1 - \gamma) \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) \quad -$$
$$\gamma \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0). \quad (14)$$

$\square$

Lemma 2. *Consider a literal l_k and clause C_j^ω. The expected payoff of including l_k in C_j^ω in the Tsetlin Machine game (Definition 2) is:*

$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) + \frac{s-1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) \quad -$$
$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1). \quad (15)$$

Proof. The expected payoff of the *Include* action can be derived from Table 5:

$$
\begin{aligned}
\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad &+ \\
\frac{s-1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) \quad &- \\
\frac{1}{s} \cdot \gamma \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) \quad &- \\
\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_j^1) \quad &- \\
\frac{1}{s} \cdot (1-\gamma) \cdot P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) \quad &- \\
\frac{1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_j^1), \quad &
\end{aligned}
\tag{16}
$$

which again simplifies to:

$$
\begin{aligned}
\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad &+ \\
\frac{s-1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) \quad &- \\
\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) \quad &- \\
\frac{1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1). \quad &
\end{aligned}
\tag{17}
$$

□

4.4 Tsetlin Automaton Convergence Criteria

Most bandit algorithms attempt to learn the action with the largest expected payoff. A Tsetlin Automaton, however, can only learn this action if the expected payoff of the action is positive (i.e., the probability of receiving a reward is larger than the probability of receiving a penalty) [3]. If the latter criterion is not fulfilled, the Tsetlin Automaton instead moves towards its center states (ϕ_N and ϕ_{N+1}), that is, towards indecision. This property is crucial for eliminating sub-optimal Nash equilibria in Tsetlin Machine learning, and is one of the reasons why it is not straightforward to replace the Tsetlin Automaton with another bandit algorithm, without loss in learning accuracy. So, in order to have a stable Nash equilibrium in Tsetlin Machine learning, each Tsetlin Automaton action in the equilibrium must not only have the largest expected payoff, the expected payoff must also be positive.

Lemma 3. *By increasing the number of states, a two-action Tsetlin Automaton converges to performing an action $\alpha_z, z \in \{0, 1\}$, with probability arbitrarily close to unity iff:*

1. *The action has positive expected payoff: $E[R|\alpha_z] > 0$.*

2. *The action has the largest expected payoff: $E[R|\alpha_z] > E[R|\alpha_{1-z}]$.*

Proof. The lemma follows trivially from the Tsetlin Automaton convergence proof found in [3].
□

4.5 Nash Equilibria Without Noise

Lemma 4. *Consider a pattern recognition problem per Definition 1 without noise, $P(y \neq \omega|X^\omega) = 1 - \gamma = 0$, and a Tsetlin Machine game per Definition 2 with a single clause $C^1(X)$ of positive polarity. The sub-patterns $Q_h^1(X)$, $h \in \{1, \ldots n/2\}$, of the pattern recognition problem are then all Nash equilibria of the game. Additionally, the expected payoff of each player action in each equilibrium is positive, hence not rejected by the Tsetlin Automata players.*

19

Proof. We first analyse the case when the *Exclude* action is optimal, before addressing the *Include* action.

Exclude is Optimal. From Property 1 and Property 2 of Definition 1, we have $P(\mathcal{X}_{jk}^1|\mathcal{X}^1) < \frac{1}{s} < P(\mathcal{X}_j^1|\mathcal{X}^1) \Leftrightarrow \frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} < \frac{1}{s} < \frac{P(\mathcal{X}_j^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)}$, which simplifies to $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} < \frac{1}{s} < \frac{P(\mathcal{X}_j^1)}{P(\mathcal{X}^1)}$ because of Property 4. From Property 4 we know in particular that $[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0 = \emptyset$. Under these conditions, we must verify that *Exclude* is also the preferred action according to Lemma 3.

We first verify that the expected payoff of *Exclude* (Eqn. 12) is positive to accommodate for Criterion 1 of Lemma 3. We here assume no noise, i.e., $\gamma = 1$, and thus can remove the addends containing the factor $(1 - \gamma)$ from Eqn. 12. We further know that $P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) = 0$ because $[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0 = \emptyset$. Hence, we can remove addends including that factor as well, ending up with:

$$\frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) \quad > \quad 0. \tag{18}$$

Because $\mathcal{X}_j^1$ is a subset of $\mathcal{X}^1$ (Property 4), we have:

$$\frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1) \quad > \quad 0. \tag{19}$$

Dividing by $P(\mathcal{X}^1)$ on each side we get:

$$\frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad > \quad 0. \tag{20}$$

With $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} < \frac{1}{s}$, we have $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} > \frac{s-1}{s}$. This clearly entails that the left side is positive.

We next verify that the expected payoff of *Exclude* is larger than that of *Include* to ensure Criterion 2 of Lemma 3. Performing the same simplifications as above, however, for Eqn. 15, we end up with the following condition:

$$\frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad < \quad 0. \tag{21}$$

For the same reasons that the expected payoff of the *Exclude* action was positive, the expected payoff of *Include* is negative, which clearly is inferior.

Include is Optimal. Conversely, given that *Include* is optimal, we have: $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} > \frac{1}{s}$ (Property 1) and $\mathcal{X}_{jk}^1 \cap \mathcal{X}^0 = \emptyset$ (from Property 4). With no noise we can again simplify the calculation of the expected payoff of *Include*, reducing Eqn. 15 to:

$$\frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad > \quad 0. \tag{22}$$

With $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} > \frac{1}{s}$ (Property 1), we must have $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} < \frac{s-1}{s}$. Accordingly, the expected payoff of *Include* is positive.

We now turn to investigating whether the expected payoff of *Include* is larger than the expected payoff of *Exclude* (Criterion 2 of Lemma 3), which we show by verifying that the expected payoff of *Exclude* is negative. We can again simplify Eqn. 12, however, this time without knowing that $[\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0 = \emptyset$:

$$\frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1) - P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) \quad < \quad 0. \tag{23}$$

Dividing by $P(\mathcal{X}^1)$ on each side we get:

$$\frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)}{P(\mathcal{X}^1)} \quad < \quad 0. \tag{24}$$

Again, with $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} > \frac{1}{s}$, we have $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} < \frac{s-1}{s}$. From this result alone, the left side is negative. Additionally, the addend $\frac{P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)}{P(\mathcal{X}^1)}$ is negative as well, reducing the expected payoff of *Exclude* even further. So, changing from *Include* to *Exclude* gives a net expected loss. $\qquad \square$

Lemma 5. *Consider a pattern recognition problem per Definition 1 without noise, $P(y \neq \omega | X^\omega) = 1 - \gamma = 0$, and a Tsetlin Machine game per Definition 2 with a single clause $C^1(X)$ of positive polarity. If clause $C^1(X)$ deviates from all of the sub-patterns $Q_h^1(X)$, $h \in \{1, \ldots n/2\}$, of the pattern recognition problem, it is either not a Nash equilibrium in the game or it is produced by an action with negative expected payoff.*

Proof. There are three kinds of sub-optimal configurations for a clause: (i) The clause is too infrequent; (ii) The clause is too frequent; or (iii) The clause outputs 1 for input vectors from $\mathcal{X}^0$. We must show that none of these fulfill both criteria of Lemma 3.

In case (i) there exists at least one literal l_k that has been included in the clause, for which $\frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} < \frac{1}{s}$. That is, less than $\frac{1}{s}$ of $\mathcal{X}^1$ is covered by the clause, in frequency. It is sufficient to show that the expected payoff of *Include* (Eqn. 15) is negative for $\gamma = 1$:

$$\frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) - \frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) \quad < \quad 0. \tag{25}$$

Dividing by $P(\mathcal{X}^1)$ on each side we get:

$$\frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} - \frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad < \quad 0. \tag{26}$$

With $\frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} < \frac{1}{s}$, we must have $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} > \frac{s-1}{s}$. We can thus conclude that the expected payoff of *Include* is negative.

We now turn to case (ii), that is, the case where the clause is too frequent. Then there must exist an excluded literal l_k such that $\frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} > \frac{1}{s}$. It is then sufficient to show that the expected payoff of *Exclude* is negative:

$$\frac{1}{s} \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - \frac{s-1}{s} \cdot \frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} - \frac{P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0)}{P(\mathcal{X}^1)} \quad < \quad 0. \tag{27}$$

With $\frac{P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} > \frac{1}{s}$, we have $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} < \frac{s-1}{s}$. Again, the left side is negative.

Apart from the sub-optimal configurations where too many or too few literals are included in the clause, we must also consider case (iii), that is, scenarios where $P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) > 0$. This means that the corresponding clause $C_j^1(X)$ outputs 1 for input vectors $X \in [\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0$. Accordingly, it outputs 1 when $P(y = 1|X) < P(y = 0|X)$, which is sub-optimal and violates Property 4 of Definition 1. It must then clearly exist a literal l_k that has been excluded from the clause and that is zero for input vectors $X \in [\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0$. If not, an optimal configuration would not exist, which it does by problem definition. Including l_k in the clause would make the clause output 0 for input vectors $X \in [\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0$. This will thus eliminate $X \in [\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0$ from the input space captured by the clause. Accordingly, we must ensure that excluding $l_k = 0$ gives a negative expected payoff:

$$\frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) - P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) \quad < \quad 0. \tag{28}$$

We can assume that $\frac{1}{s} \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)$ and $\frac{s-1}{s} \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1)$ are roughly equal. If they are not, we have already determined in the previous two paragraphs that the resulting clause is not a Nash equilibrium. Thus, what remains is to show that $P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) > 0$, which is stated by case (iii) itself. $\qquad \square$

Theorem 1. *Consider a pattern recognition problem per Definition 1 without noise, $P(y \neq \omega | X^\omega) = 1 - \gamma = 0$, and a Tsetlin Machine game per Definition 2 with a single clause $C^1(X)$ of positive polarity. The sub-patterns $Q_h^1(X)$, $h \in \{1, \dots n/2\}$, are then the only Nash equilibria of the game, where the expected payoff of each player action is positive, hence not rejected by the Tsetlin Automata players.*

Proof. This theorem follows from Lemma 3, Lemma 4 and Lemma 5. □

4.6 Nash Equilibria with Static Noise

Lemma 6. *Consider a pattern recognition problem per Definition 1 with static noise, $P(y \neq \omega | X^\omega) = 1 - \gamma, 0.5 < \gamma < 1.0$, and output classes of equal size, $P(X^0) = P(X^1) = 0.5$. Consider further a Tsetlin Machine game per Definition 2 with a single clause $C^1(X)$ of positive polarity. Each sub-pattern $Q_h^1(X)$, $h \in \{1, \dots n/2\}$, of the pattern recognition problem is a Nash equilibrium in the game when s is scaled by a factor t,*

$$\frac{b_1}{\gamma} \; < t < \; \frac{b_2}{\frac{b_2}{b_3}(1 - \gamma) + 2(\gamma - 0.5)}, \tag{29}$$

where, $b_2 = \min_{j,k} \frac{P(\mathcal{X}^1)}{P(\mathcal{X}_{jk}^1) \cdot s}, b_2 > 1$, and $b_3 = \max_{j,k} \frac{P(\mathcal{X}^1)}{P(\mathcal{X}_j^1) \cdot s}, 0 < b_3 < 1$, define a worst-case upper bound for t when Exclude is optimal. Further, $b_1 = \max_{j,k} \frac{P(\mathcal{X}^1)}{P(\mathcal{X}_{jk}^1) \cdot s}, 0 < b_1 < 1$, defines a worst-case lower-bound when Include is optimal. Additionally, the expected payoff of each player action in each equilibrium is positive, hence not rejected by the Tsetlin Automata players.

As seen, increasing noise (reduced γ) tightens the lower bound, but loosens the upper. Further, an increasing b_2 loosens the upper bound, while a decreasing b_1 loosens the lower bound.

Proof. We first analyse the case when *Exclude* is optimal, before addressing *Include*.

Exclude is Optimal. Since *Exclude* is optimal, we have (follows from Definition 1): $P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) = 0$, $P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^0) = 0$, $P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) = P(\mathcal{X}_{jk}^1)$, $P([\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1] \cap \mathcal{X}^1) = P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1)$, and $P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1) = P(\mathcal{X}^0)$. We consider Criterion 1 of Lemma 3 first, i.e., that the expected payoff from Eqn. 12 is positive, which simplifies to:

$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) + \frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0) \quad -$$
$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1) - (1 - \gamma) \cdot P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1) \quad > \quad 0. \tag{30}$$

Dividing by $P(\mathcal{X}^1)$ on each side, we get:

$$\frac{1}{s} \cdot \gamma \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} + \frac{1}{s} \cdot (1 - \gamma) \cdot \frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)} \quad -$$
$$\frac{s-1}{s} \cdot \gamma \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - (1 - \gamma) \cdot \frac{P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad > \quad 0. \tag{31}$$

Clearly, to maintain positive expected payoff after the introduction of static noise, we need to compensate for the addends introduced by the noise, i.e., the $(1 - \gamma)$-addends. To this end, we artificially scale up the specificity parameter s by a factor t. Simplifying further, we replace $\frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)}$ with 1:

$$\frac{1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} + \frac{1}{ts} \cdot (1 - \gamma) \quad -$$
$$\frac{ts-1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} - (1 - \gamma) \cdot \frac{P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} \quad > \quad 0. \tag{32}$$

Since we consider the case where excluding literal l_k is optimal, we have $b_2 > 1$ such that: $\frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} = \frac{1}{b_2 s} \Rightarrow b_2 = \frac{P(\mathcal{X}^1)}{P(\mathcal{X}_{jk}^1) \cdot s}$. That is, b_2 is an unknown factor that measures exactly how much including l_k constrains the clause beyond $\frac{1}{s}$, according to Property 1 and Property 2 of Definition 1. This in turn, means that $\frac{P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} = 1 - \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} = \frac{b_2 s - 1}{b_2 s}$. We finally introduce $b_3, 0 < b_3 < 1$. The role of b_3 is to exactly determine how much excluding literal l_k loosens up the clause beyond $\frac{1}{s}$: $\frac{P(\mathcal{X}_j^1)}{P(\mathcal{X}^1)} = \frac{1}{b_3 s} \Rightarrow b_3 = \frac{P(\mathcal{X}^1)}{P(\mathcal{X}_j^1) \cdot s}$. This leads to $\frac{P(\mathcal{X}_j^1 \setminus \mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} = \frac{P(\mathcal{X}_j^1)}{P(\mathcal{X}^1)} - \frac{P(\mathcal{X}_{jk}^1)}{P(\mathcal{X}^1)} = \frac{1}{b_3 s} - \frac{1}{b_2 s}$. With these amendments, we revisit the condition that the expected payoff of *Exclude* must be positive:

$$\frac{1}{ts} \cdot \gamma \cdot \frac{b_2 s - 1}{b_2 s} + \frac{1}{ts} \cdot (1 - \gamma) - \frac{ts - 1}{ts} \cdot \gamma \cdot \frac{1}{b_2 s} - (1 - \gamma) \cdot \left(\frac{1}{b_3 s} - \frac{1}{b_2 s} \right) > 0. \quad (33)$$

Through a sequence of standard algebraic operations, the above condition can be simplified as follows:

$$\gamma \cdot \frac{b_2 s - 1}{b_2 s} + (1 - \gamma) - (ts - 1) \cdot \gamma \cdot \frac{1}{b_2 s} - ts \cdot (1 - \gamma) \cdot \left(\frac{1}{b_3 s} - \frac{1}{b_2 s} \right) > 0 \quad (34)$$

$$\gamma \cdot \frac{b_2 s - 1}{b_2 s} + (1 - \gamma) - (ts - 1) \cdot \gamma \cdot \frac{1}{b_2 s} - ts \cdot (1 - \gamma) \cdot \frac{b_2 - b_3}{b_2 b_3 s} > 0 \quad (35)$$

$$\gamma \cdot (b_2 s - 1) + b_2 s \cdot (1 - \gamma) - (ts - 1) \cdot \gamma - ts \cdot (1 - \gamma) \cdot \frac{b_2 - b_3}{b_3} > 0 \quad (36)$$

$$b_2 s \gamma - \gamma + b_2 s - b_2 s \gamma - ts \gamma + \gamma - ts \cdot (1 - \gamma) \cdot \frac{b_2 - b_3}{b_3} > 0 \quad (37)$$

$$b_2 \gamma + b_2 - b_2 \gamma - t \gamma - t \cdot (1 - \gamma) \cdot \frac{b_2 - b_3}{b_3} > 0 \quad (38)$$

$$b_2 \gamma + b_2 - b_2 \gamma - t \gamma - t \cdot (1 - \gamma) \cdot \left(\frac{b_2}{b_3} - 1 \right) > 0 \quad (39)$$

$$b_2 \gamma + b_2 - b_2 \gamma - t \gamma - \left(t \cdot (1 - \gamma) \cdot \frac{b_2}{b_3} - t \cdot (1 - \gamma) \right) > 0 \quad (40)$$

$$b_2 \gamma + b_2 - b_2 \gamma - t \gamma - t \cdot (1 - \gamma) \cdot \frac{b_2}{b_3} + t \cdot (1 - \gamma) > 0 \quad (41)$$

$$b_2 - t \gamma - t \cdot (1 - \gamma) \cdot \frac{b_2}{b_3} + t \cdot (1 - \gamma) > 0 \quad (42)$$

$$b_2 - t \gamma - \frac{b_2}{b_3} t + \frac{b_2}{b_3} t \gamma + t - t \gamma > 0 \quad (43)$$

$$b_2 - t \left(\gamma + \frac{b_2}{b_3} - \frac{b_2}{b_3} \gamma - 1 + \gamma \right) > 0 \quad (44)$$

$$\frac{b_2}{\gamma + \frac{b_2}{b_3} - \frac{b_2}{b_3} \gamma - 1 + \gamma} > t \quad (45)$$

$$\frac{b_2}{\frac{b_2}{b_3}(1 - \gamma) + 2(\gamma - \frac{1}{2})} > t \quad (46)$$

This means that it is possible to increase t up to a certain boundary, weakening reinforcement of *Exclude*, without making the expected payoff of *Exclude* turn negative.

We now show that the expected payoff of *Include* is negative, fulfilling Criterion 2 of Lemma 3:

$$\frac{s - 1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^1) + \frac{s - 1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}_{jk}^1 \cap \mathcal{X}^0) -$$

$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}_{jk}^1) < 0. \quad (47)$$

Since *Exclude* is optimal, we can simplify as follows (per Definition 1):

$$\frac{s - 1}{s} \cdot \gamma \cdot P(\mathcal{X}_{jk}^1) - \frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}_{jk}^1) - \frac{1}{s} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0) < 0. \quad (48)$$

Introducing the scaling factor $t > 1$ for s and dividing by $P(\mathcal{X}^1)$ we get:

$$\frac{ts-1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} - \frac{1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} - \frac{1}{ts} \cdot (1-\gamma) \cdot \frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)} \quad < \quad 0. \tag{49}$$

Because we have established that Eqn. 32 is positive, it follows that the above equation is negative. This is due to the fact Eqn. 32 is still positive after removing the last addend, and the above equation is simply the negated version of the latter.

Include is Optimal. To fulfill Criterion 1 of Lemma 3 we must show that the expected payoff of *Include* is positive:

$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}^1_{jk} \cap \mathcal{X}^1) + \frac{s-1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^1_{jk} \cap \mathcal{X}^0) \quad -$$
$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk}) - \frac{1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^0 \setminus \mathcal{X}^1_{jk}) \quad > \quad 0. \tag{50}$$

Since *Include* is optimal, we have: $P(\mathcal{X}^1_{jk} \cap \mathcal{X}^0) = 0$, $P(\mathcal{X}^1_{jk} \cap \mathcal{X}^1) = P(\mathcal{X}^1_{jk})$ and $P(\mathcal{X}^0 \setminus \mathcal{X}^1_{jk}) = P(\mathcal{X}^0)$, again per Property 4 of Definition 1. Accordingly, we can simplify the above expression to:

$$\frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}^1_{jk}) - \frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk}) - \frac{1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^0) \quad > \quad 0. \tag{51}$$

To compensate for the addends introduced by noise, we again increase the s-parameter by artificially multiplying it with a factor $t > 1$. Dividing by $P(\mathcal{X}^1)$ we get:

$$\frac{ts-1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} - \frac{1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} - \frac{1}{ts} \cdot (1-\gamma) \cdot \frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)} \quad > \quad 0. \tag{52}$$

Now we introduce $0 < b_1 < 1$, which tells exactly how much larger $\frac{P(\mathcal{X}^1_{jk})}{P(\mathcal{X}^1)}$ is compared to $\frac{1}{s}$, i.e., $\frac{P(\mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} = \frac{1}{b_1 s} \Rightarrow b_1 = \frac{P(\mathcal{X}^1)}{P(\mathcal{X}^1_{jk}) \cdot s}$ as determined by Property 1 and Property 2 of Definition 1, we can simplify the condition as follows, also replacing $\frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)}$ with 1:

$$\frac{ts-1}{ts} \cdot \gamma \cdot \frac{1}{b_1 s} - \frac{1}{ts} \cdot \gamma \cdot \frac{b_1 s - 1}{b_1 s} - \frac{1}{ts} \cdot (1-\gamma) \quad > \quad 0 \tag{53}$$

$$\frac{ts-1}{t} \cdot \gamma \cdot \frac{1}{b_1} - \frac{1}{t} \cdot \gamma \cdot \frac{b_1 s - 1}{b_1} - \frac{s}{t} \cdot (1-\gamma) \quad > \quad 0 \tag{54}$$

$$(ts-1) \cdot \gamma - \gamma \cdot (b_1 s - 1) - b_1 s \cdot (1-\gamma) \quad > \quad 0 \tag{55}$$

$$ts\gamma - \gamma - b_1 s \gamma + \gamma - b_1 s + b_1 s \gamma \quad > \quad 0 \tag{56}$$

$$ts\gamma - b_1 s \quad > \quad 0 \tag{57}$$

$$\frac{b_1}{\gamma} \quad < \quad t \tag{58}$$

This leads to the following range for t, recovering a Nash equilibrium corresponding to the sub-patterns of the pattern recognition problem:

$$\frac{b_1}{\gamma} \quad < \quad t \quad < \quad \frac{b_2}{\frac{b_2}{b_3}(1-\gamma) + 2(\gamma - \frac{1}{2})} \tag{59}$$

That is, scaling s by a factor t allows us to recover from some degree of static noise, as governed by b_1, b_2, and b_3.

We finally need to verify that the expected payoff of *Exclude* is negative, in order to fulfill Criterion 2 of Lemma 3:

$$\frac{1}{s} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk}) + \frac{1}{s} \cdot (1-\gamma) \cdot P(\mathcal{X}^0) - \frac{s-1}{s} \cdot \gamma \cdot P(\mathcal{X}^1_{jk}) \quad -$$
$$(1-\gamma) \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^1) - \gamma \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0) \quad < \quad 0. \tag{60}$$

Again, introducing the scaling factor $t > 1$ for s and dividing by $P(\mathcal{X}^1)$ we get:

$$\frac{1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} + \frac{1}{ts} \cdot (1 - \gamma) \cdot \frac{P(\mathcal{X}^0)}{P(\mathcal{X}^1)} - \frac{ts-1}{ts} \cdot \gamma \cdot \frac{P(\mathcal{X}^1_{jk})}{P(\mathcal{X}^1)} \quad -$$

$$(1 - \gamma) \cdot \frac{P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^1)}{P(\mathcal{X}^1)} - \gamma \cdot \frac{P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0)}{P(\mathcal{X}^1)} \quad < \quad 0. \qquad (61)$$

Since we have established that Eqn. 52 is positive, it follows that the above equation is negative. This is due to the above equation simply being the negated version of the latter, subtracting two extra addends, clearly making it negative. $\qquad \square$

Lemma 7. *Consider a pattern recognition problem per Definition 1 with static noise, $P(y \neq \omega|X^\omega) = 1 - \gamma, 0.5 < \gamma < 1.0$, and output classes of equal size, $P(y = 0) = P(y = 1) = 0.5$. Consider further a Tsetlin Machine game per Definition 2 with only a single clause $C^1(X)$ of positive polarity. If clause $C^1(X)$ deviates from all of the sub-patterns $Q^1_h(X), h \in \{1, \ldots n/2\}$, of the pattern recognition problem, it is either not a Nash equilibrium in the game or it is produced by an action with negative payoff.*

Proof. For too infrequent or too frequent clauses, the proof follows the structure of the corresponding proof for the noise-free cases (i) and (ii). However, we need to introduce the scaling factor t, as demonstrated above.

Case (iii) requires some more scrutiny. That is, we need to show that when $P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0) > 0$, the expected payoff of *Exclude* is negative:

$$\frac{1}{ts} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk}) \quad +$$

$$\frac{1}{ts} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0) \quad -$$

$$\frac{ts-1}{ts} \cdot \gamma \cdot P(\mathcal{X}^1_{jk}) \quad -$$

$$(1 - \gamma) \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^1) \quad -$$

$$\gamma \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0) \quad < \quad 0. \qquad (62)$$

Again we can assume that $\frac{ts-1}{ts} \cdot \gamma \cdot P(\mathcal{X}^1_{jk}) - \frac{1}{ts} \cdot \gamma \cdot P(\mathcal{X}^1 \setminus \mathcal{X}^1_{jk}) - \frac{1}{ts} \cdot (1 - \gamma) \cdot P(\mathcal{X}^0)$ is close to zero. Otherwise, we did not have a Nash equilibrium in the first place, due to case (i) and case (ii). Thus, we only need to verify that:

$$(1 - \gamma) \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^1) + \gamma \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0) \quad > \quad 0.$$

The above condition is clearly true because we have $P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^0) > 0$ by case (iii) itself. Additionally, because we have noise, $(1 - \gamma) \cdot P([\mathcal{X}^1_j \setminus \mathcal{X}^1_{jk}] \cap \mathcal{X}^1)$ is strictly positive as well. $\quad \square$

Theorem 2. *Consider a pattern recognition problem per Definition 1 with static noise, $P(y \neq \omega|X^\omega) = 1 - \gamma, 0.5 < \gamma < 1.0$, and output classes of equal size, $P(y = 0) = P(y = 1) = 0.5$. Consider further a Tsetlin Machine game per Definition 2 with only a single clause $C^1(X)$ of positive polarity. The sub-patterns $Q^1_h(X), h \in \{1, \ldots n/2\}$, of the pattern recognition problem are then the only Nash equilibria of the game, where the expected payoff of each player action is positive, hence not rejected by the Tsetlin Automata players.*

Proof. This theorem follows from Lemma Lemma 3, Lemma 6, and Lemma 7. $\qquad \square$

4.7 Coordination of Multiple Tsetlin Automata Teams

Theorem 3. *Consider a pattern recognition problem per Definition 1 without noise, $P(y \neq \omega|X^\omega) = 1 - \gamma = 0$, and a Tsetlin Machine game per Definition 2 with $T \cdot n$ clauses. Then the only Nash equilibrium of the Tsetlin Machine game is a configuration where T Tsetlin Machine clauses $C^\omega_j, \omega \in \{0, 1\}, j \in \{1, \ldots, T \cdot n/2\}$, duplicate each sub-pattern $Q^\omega_h, \omega \in \{0, 1\}, h \in \{1, \ldots, n/2\}$.*

Proof. We already known from Theorem 1 that if a clause C_j^ω is not duplicating any of the sub-patterns Q_h^ω, the Tsetlin Automata of the clause are not in a Nash Equilibrium. We therefore only consider the case where all clauses C_j^ω are duplicating some sub-pattern Q_h^ω. Let us first consider positive polarity clauses ($\omega = 1$). Assume that more than T of these clauses duplicate a particular sub-pattern Q_h^1. Then, clearly, there are fewer than T clauses duplicating some other sub-pattern $Q_{h'}^1, h' \neq h$. The Tsetlin Automata that take part in duplicating Q_h^1 will then only receive Inaction-feedback when facing sub-pattern Q_h^1, as seen from Eqn. 8. However, when they observe sub-pattern $Q_{h'}^1$ they will receive Type I Feedback with probability larger than 0. Type I Feedback pulls the Tsetlin Automata away from Q_h^1 and towards $Q_{h'}^1$. Hence, we do not have a Nash equilibrium. For the same reasons, we do not have a Nash equilibrium when more than T negative polarity clauses capture a sub-pattern Q_h^0 due to Eqn. 9. When there are T clauses per sub-pattern Q_h^ω, on the other hand, the whole system is in equilibrium due to Theorem 1 and Eqns. 8-11. $\square$

The proof for static noise follows the above structure, however, relies on Theorem 2 instead. The proof is left out here for the sake of brevity.

Remark 1. Note that a Tsetlin Machine in practice also learns robustly when the total number of clauses available is less than $T \cdot n$. This is because of the gradually increasing probability of receiving feedback the farther away one is from $T/-T$, which leads to a load balancing effect among the clauses. This effect dynamically distributes the clauses among the available sub-patterns.

Remark 2. A traditional Learning Automata approach to pattern recognition would use classification accuracy as payoff (utility) function, to give feedback to the Learning Automata. In general, this leads to the Vanishing Signal to Noise Ratio Problem described in Section 1.3. This is due to the relatively small effect each single Learning Automaton has on overall classification accuracy, and due to the noise introduced by the random behaviour of the team as a whole. In the Tsetlin Machine game, on the other hand, each single automaton gets feedback directly from the value of its literal and the value of its clause, as laid out in Table 2 and Table 3. We will now show that this local feedback scheme can be a generalized ordinal potential game [30], with classification accuracy being the global payoff (utility) function.

Theorem 4. *Let M refer to the payoff matrix of the Tsetlin Machine game from Definition 2. Further, let a_i' and a_i'' refer to the two actions available to a single automaton, while a_{-i} refers to the remaining actions in the action configuration $\overline{\alpha}$. Finally, let the summation target T be 1, only considering positive clauses. The resulting Tsetlin Machine game is a generalized ordinal potential game with regards to classification accuracy, under the constraints of Definition 1. That is, we have: $P(y = \hat{y}; a_i', a_{-i}) - P(y = \hat{y}; a_i'', a_{-i}) > 0 \Rightarrow M(a_i', a_{-i}) - M(a_i'', a_{-i}) > 0$.*

Proof. Classification accuracy is clearly maximised only when $C_j^1, j \in \{1, \ldots, n/2\}$, are duplicating $Q_h^1, h \in \{1, \ldots, n/2\}$. Any mismatch will either produce false positive output if the Tsetlin Machine clauses C_j^1 are missing literals, or false negative output if not all of the sub-patterns Q_h^1 are completely covered. Thus, during learning, there are two ways classification accuracy can increase by switching action from a_i'' to a_i': either by a decrease in the rate of false negatives, $P(y = 1, \hat{y} = 0)$, or by a decrease in the rate of false positives, $P(y = 0, \hat{y} = 1)$.

Let us first consider decreasing $P(y = 1, \hat{y} = 0)$. This can only be achieved by turning false negative output into true positive output. Clearly, for a particular input X, we have false negative output if there exists an h such that $Q_h^1(X) = 1$, with $C_j^1(X) = 0$ for all j. Let L_j^1 be the literals of clause C_j^1 and K_h^1 be the literals of Q_h^1. We can correct the false negative error *if and only if* we have a j and k so that $L_j^1 \setminus \{l_k\} \subseteq K_h^1, l_k \notin K_h^1$, implying $l_k = 0$. The false negative error can then be corrected by excluding l_k from clause C_j^1. Examining Feedback Type I for $C_j^1(X) = 0, l_k = 0$, in Table 2, which applies for $y = 1$, we observe that the expected payoff of excluding l_k is larger that the expected payoff of including it.

We consider decreasing $P(y = 0, \hat{y} = 1)$ next. This can be achieved by turning false positive output into true negative output. Clearly, for a particular input X, we have false positive output *if and only if* there exists a j such that $C_j^1(X) = 1$, with $Q_h^1(X) = 0$ for all h. Let L_j^1 be the literals of clause C_j^1 and K_h^1 be the literals of Q_h^1. We can correct the false positive output *if and only if* there exists *only one* j such that $C_j^1(X) = 1$, and if this particular $C_j^1(X)$ has left out a literal l_k of value 0, $l_k = 0, l_k \notin L_j^1$. Then the false positive output can be corrected by including $l_k = 0$ in C_j^1. Examining Feedback Type II for $C_j^1(X) = 1, l_k = 0$, in Table 3, which applies for $y = 0$, we observe that the expected payoff of including l_k is larger that the expected payoff of excluding it. $\qquad\square$

Remark 4. The ordinal potential game is *generalized* because the Tsetlin Machine game matrix M rewards action switches that have no impact on classification accuracy. That is, we do not have: $M(a_i', a_{-i}) - M(a_i'', a_{-i}) > 0 \Rightarrow P(y = \hat{y}; a_i', a_{-i}) - P(y = \hat{y}; a_i'', a_{-i}) > 0$. In particular, consider false negative output that cannot be corrected by excluding a single literal from a single clause. Yet, M still rewards excluding literals of value $0, l_k = 0$, when $C_j^1(X) = 0$, *eventually* correcting the false negative output. Similarly, when multiple clauses output 1, producing a false positive output, no single *Include* action can alone correct the false positive output. Yet, including literals of value 0 is still rewarded in the game matrix M, again, *eventually* fixing the false positive output. Thus, in this sense, the Tsetlin Machine game matrix has a longer planning horizon than just greedily optimizing classification accuracy in single steps. Note that these latter dynamics do not produce any additional Nash equilibria, as established by Theorem 3. This means that there is a one-to-one correspondence between the equilibria in the potential game and in optimizing classification accuracy.

Remark 5. As summarized in [3], the behaviour of finite state Learning Automata collectives have been studied by several researchers, showing that rational behaviour can be obtained in certain cases, when memory is infinite and the game's payoff matrix is constrained [31, 15, 32, 33, 34]. Proofs for variable structure Learning Automata are more general, on the other hand. That is, for identical payoff games, the Learning Automata collective converges to a Nash equilibrium with probability arbitrarily close to unity [3]. Further, for zero-sum games with a saddle point, there exists Learning Automata collectives that can obtain the Von Neumann value [35, 36]. In the next section, we study the behaviour of Tsetlin Machine games empirically on various datasets, including the artificial Noisy XOR dataset where the optimal classification accuracy and the corresponding Nash equilibria are known.

5 Empirical Results

In this section, we evaluate the Tsetlin Machine empirically using five datasets:

- **Binary Iris Dataset.** This is the classical Iris Dataset, however, with features in binary form.

- **Binary Digits Dataset.** This is the classical digits dataset, again with features in binary form.

- ***Axis & Allies* Board Game Dataset.** This new dataset involves optimal move prediction in a minimalistic, yet intricate, mini-game from the *Axis & Allies* board game.

- **Noisy XOR Dataset with Non-informative Features.** This artificial dataset is designed to reveal particular "blind zones" of pattern recognition algorithms. The dataset captures the renowned XOR-relation. Furthermore, the dataset contains a large number of random non-informative features to measure susceptibility towards the curse of dimensionality [37]. To examine robustness towards noise we have further randomly inverted 40% of the outputs.

- **MNIST Dataset.** The MNIST dataset is a larger scale dataset used extensively to benchmark machine learning algorithms. We have included this dataset to investigate the scalability of the Tsetlin Machine, as well as the behaviour of longer learning processes.

For these datasets, we form ensembles of 50 to 1000 independent replications with different random number streams. We do this to minimize the variance of the reported results and to provide the foundation for a statistical analysis of the merits of the different schemes evaluated.

Together with the Tsetlin Machine, we also evaluate several classical machine learning techniques using the same random number streams. This includes Multilayer Perceptron Networks, the Naive Bayes Classifier, Support Vector Machines, and Logistic Regression. Where appropriate, the different schemes are optimized by means of relatively light hyper-parameter grid searches. As an example, Figure 7 captures the impact the s parameter of the Tsetlin Machine has on mean accuracy, for the Noisy XOR Dataset. Each point in the plot measures the mean accuracy of 100 different replications of the XOR-experiment for a particular value of s. Clearly, accuracy increases with s up to a certain point, before it degrades gradually. Based on the plot, for the Noisy XOR-experiment, we decided to use an s value of 3.9.
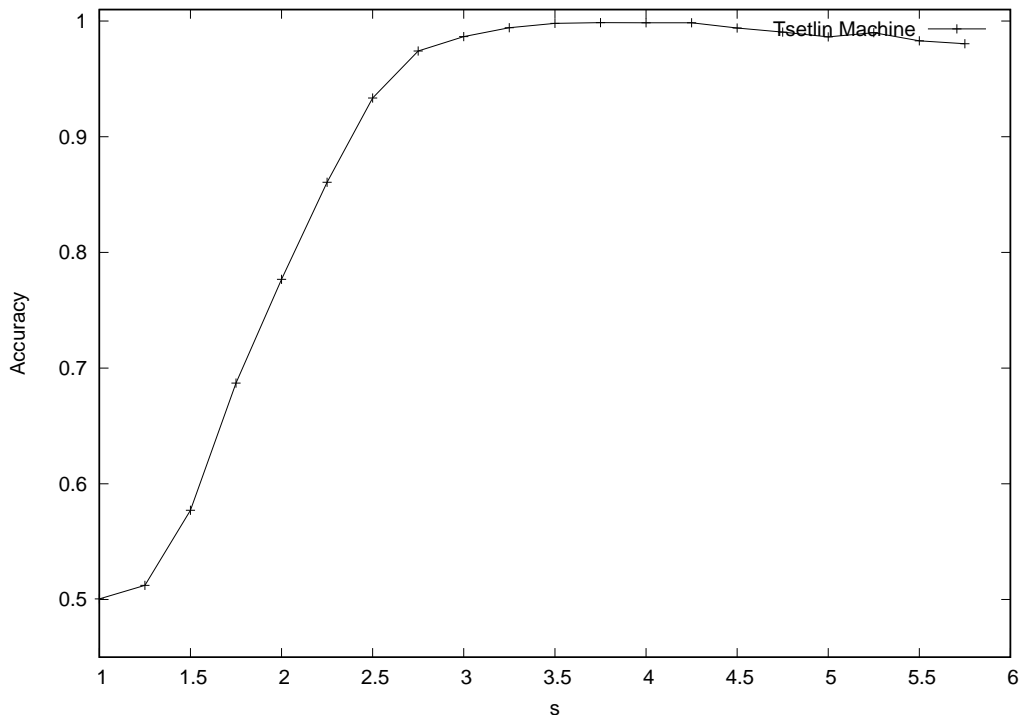

Figure 7: The mean accuracy of the Tsetlin Machine (y-axis) on the Noisy XOR Dataset for different values of the parameter s (x-axis).

5.1 The Binary Iris Dataset

We first evaluate the Tsetlin Machine on the classical Iris dataset[2]. This dataset consists of 150 examples with four inputs (Sepal Length, Sepal Width, Petal Length and Petal Width), and three possible outputs (Setosa, Versicolour, and Virginica).

We increase the challenge by transforming the four input values into one consecutive sequence of 16 bits, four bits per float. It is thus necessary to also learn how to segment the 16

---

[2] UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/iris].

bits into four partitions, and extract the numeric information. We refer to the new dataset as the Binary Iris Dataset.

We partition this dataset into a training set and a test set, with 80 percent of the data being used for training. We here randomly produce 1000 training and test data partitions. For each ensemble, we also randomly reinitialize the competing algorithms, to gain information on stability and robustness. The results are reported in Table 6.

Technique/Accuracy (%)	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	95.0 ± 0.2	86.7	100.0	80.0	100.0
Naive Bayes	91.6 ± 0.3	83.3	96.7	70.0	100.0
Logistic Regression	92.6 ± 0.2	86.7	100.0	76.7	100.0
Multilayer Perceptron Networks	93.8 ± 0.2	86.7	100.0	80.0	100.0
SVM	93.6 ± 0.3	86.7	100.0	76.7	100.0

Table 6: The Binary Iris Dataset – accuracy on test data.

The Tsetlin Machine[3] used here employs 300 clauses, and uses an s-value of 3.0 and a summation target T of 10. Furthermore, the individual Tsetlin Automata each has 100 states. This Tsetlin Machine is run for 500 epochs, and it is the accuracy after the final epoch that is reported. Propositional formulas with higher test accuracy are often found in preceding epochs because of the random exploration of the Tsetlin Machine. However, to avoid overfitting to the test set by handpicking the best configuration found, we instead simply use the last configuration produced.

In Table 6, we list mean accuracy with 95% confidence intervals, 5 and 95 percentiles, as well as the minimum and maximum accuracy obtained, across the 1000 experiment runs we executed. As seen, the Tsetlin Machine provides the highest mean accuracy. For the 95 %ile scores, however, most of the schemes obtain 100% accuracy. This can be explained by the small size of the test set, which merely contains 30 examples. Thus it is easier to stumble upon a random configuration that happens to provide fault-free classification. Since the test set is merely a sample of the corresponding real-world problem, it is reasonable to assume that higher mean accuracy translates to more robust performance overall.

The training set, on the other hand, reveals subtler differences between the schemes. The results obtained on the training set are shown in Table 7. As seen, the SVM here provides the highest mean accuracy, while the Tsetlin Machine provides the second highest. However, the large drop in accuracy from the training data to the test data for the SVM indicates overfitting on the training data.

Technique	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	96.6 ± 0.05	95.0	98.3	94.2	99.2
Naive Bayes	92.4 ± 0.08	90.0	94.2	85.8	97.5
Logistic Regression	93.8 ± 0.07	92.5	95.8	90.0	97.5
Multilayer Perceptron Network	95.0 ± 0.07	93.3	96.7	92.5	98.3
SVM	96.7 ± 0.05	95.8	98.3	95.8	99.2

Table 7: The Binary Iris Dataset – accuracy on training data.

5.2 The Binary Digits Dataset

We next evaluate the Tsetlin Machine on the classical Pen-Based Recognition of Handwritten Digits Dataset[4]. The original dataset consists of 250 handwritten digits from 44 different writers, for a total number of 10992 instances. We increase the challenge by removing the

---

[3]In this experiment, we use a Multi-Class Tsetlin Machine, described in Section 6.1. We also apply Boosting of True Positive Feedback to *Include* actions as described in Section 3.3.3.

[4]UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits].

individual pixel value structure, transforming the 64 different input features into a sequence of 192 bits, 3 bits per pixel. We refer to the modified dataset as the Binary Digits Dataset. Again we partition the dataset into training and test sets, keeping 80 percent of the data for training.

The Tsetlin Machine[5] used here contains 1000 clauses, and uses a specificity s of 3.0 and a summation target T of 10. Furthermore, the individual Tsetlin Automata each has 1000 states. The Tsetlin Machine is run for 300 epochs, and it is the accuracy after the final epoch that is reported.

Table 8 reports mean accuracy with 95% confidence intervals, 5 and 95 percentiles, as well as the minimum and maximum accuracy obtained, across the 100 experiment runs we executed. As seen, the Tsetlin Machine again clearly provides the highest accuracy on average, also when taking the 95% confidence intervals into account. For this dataset, the Tsetlin Machine is also superior when it comes to the maximal accuracy found across the 100 replications of the experiment, as well as for the 95 %ile results.

Technique/Accuracy (%)	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	95.7 ± 0.2	93.9	97.2	92.5	98.1
Naive Bayes	91.3 ± 0.3	88.9	93.6	87.2	94.4
Logistic Regression	94.0 ± 0.2	91.9	95.8	90.8	96.9
Multilayer Perceptron Network	93.5 ± 0.2	91.7	95.3	90.6	96.7
SVM	50.5 ± 2.2	30.3	67.4	25.8	77.8

Table 8: The Binary Digits Dataset – accuracy on test data.

Performing poor on the test data and well on the training data indicates susceptibility to overfitting. Table 9 reveals that the other techniques, apart from the Naive Bayes Classifier, perform significantly better on the training data, unable to transfer this performance to the test data.

Table 1 visualizes one of the clauses produced by the Tsetlin Machine, capturing handwritten digit '1'. As seen, the pattern should be relatively easy to interpret for humans compared to, e.g., a neural network.

Technique	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	100.0 ± 0.01	99.9	100.0	99.8	100.0
Naive Bayes	92.9 ± 0.07	92.4	93.5	91.3	93.7
Logistic Regression	99.6 ± 0.02	99.4	99.7	99.3	99.9
Multilayer Perceptron Network	100.0 ± 0.0	100.0	100.0	100.0	100.0
SVM	100.0 ± 0.0	100.0	100.0	100.0	100.0

Table 9: The Binary Digits Dataset – accuracy on training data.

5.3   The *Axis & Allies* Board Game Dataset

Besides the two classical datasets, we also have built a new dataset based on the board game *Axis & Allies*[6]. We designed this dataset to exhibit intricate pattern structures, involving optimal move prediction in a subgame of *Axis & Allies*. In *Axis & Allies*, every piece on the board are potentially moved each turn. Additionally, new pieces are introduced throughout the game, as a result of earlier decisions. This arguably yields a larger search tree than the ones we find in Go and chess. Finally, the outcome of battles are determined by dice, rendering the game stochastic.

The *Axis & Allies* Board Game Dataset consists of 10 000 board game positions, exemplified in Figure 8. Player 1 owns the "Caucasus" territory in the figure, while Player 2 owns "Ukraine"

---

[5]In this experiment, we used a Multi-Class Tsetlin Machine, described in Section 6.1. We also apply Boosting of True Positive Feedback to *Include* actions as described in Section 3.3.3.

[6]http://avalonhill.wizards.com/games/axis-and-allies

and "West Russia". At start-up, each player is randomly assigned 0-10 tanks and 0-20 infantry each. These units are the starting forces. For Player 2, the units are randomly distributed among his two territories. The game consists of two rounds. First Player 1 attacks. This is followed by a counter attack by Player 2. In order to win, Player 1 needs to capture both of "Ukraine" and "West Russia". Player 2, on the other hand, only needs to take "Caucasus".

At Start						Optimal Attack			
Caucasus		W. Russia		Ukraine		W. Russia		Ukraine	
Inf	Tnk	Inf	Tnk	Inf	Tnk	Inf	Tnk	Inf	Tnk
16	4	11	4	5	4	0	0	3	4
19	3	6	1	6	3	7	2	12	1
9	1	1	3	0	5	0	0	0	0

Table 10: The *Axis & Allies* Board Game Dataset.

To produce the dataset, we built an *Axis & Allies* Board Game simulator. This allowed us to find the optimal attack for each assignment of starting forces. The resulting input and output variables are shown in Table 10. The at start forces are to the left, while the optimal attack forces can be found to the right. In the first row, for instance, it is optimal for Player 1 to launch a preemptive strike against the armor in Ukraine (armor is better offensively than defensively), to destroy offensive power, while keeping the majority of forces for defense.

We use 25% of the data for training, and 75% for testing, randomly producing 100 different partitions of the dataset. The Tsetlin Machine employed here contains 10 000 clauses, and uses an s-value of 40.0 and a summation target T of 10. Furthermore, the individual Tsetlin Automata each has 1000 states. The Tsetlin machine is run for 200 epochs, and it is the accuracy after the final epoch that is reported.

Table 11 reports the results from predicting output bit 5 among the 20 output bits (as representative for all of the bits). In the table, we list mean accuracy with 95% confidence intervals, 5 and 95 percentiles, as well as the minimum and maximum accuracy obtained, across the 100 experiment runs we executed. As seen in the table, apparently only the Tsetlin

Technique/Accuracy (%)	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	87.7 ± 0.0	87.4	88.0	87.2	88.1
Naive Bayes	80.1 ± 0.0	80.1	80.1	80.1	80.1
Logistic Regression	77.7 ± 0.0	77.7	77.7	77.7	77.7
Multilayer Perceptron Network	87.6 ± 0.1	87.1	88.1	86.6	88.3
SVM	83.7 ± 0.0	83.7	83.7	83.7	83.7
Random Forest	83.1 ± 0.1	82.3	83.8	81.6	84.1

Table 11: The *Axis & Allies* Dataset – accuracy on test data.

Machine and the neural network are capable of properly handling the complexity of the dataset, providing statistically similar performance. The Tsetlin Machine is quite stable performance-wise, while the neural network performance varies more.

However, the number of clauses needed to achieve the above performance is quite high for the Tsetlin Machine, arguably due to its flat one-layer architecture. Another reason that can explain the need for a large number of clauses can be the intricate nature of the mini-game of Axis & Allies. Since we need an s-value as large as 40, clearly, some of the pertinent sub-patterns must be quite fine-grained. Because the s-value is global, all patterns, even the coarser ones, must be learned at this fine granularity. A possible next step in the research on the Tsetlin Machine could therefore be to investigate the effect of having clauses with different s-values – some with smaller values for the rougher patterns, and some with larger values for the finer patterns.

As a final observation, Table 12 reports performance on the training data. Random Forest distinguishes itself by almost perfect predictions for the training data, thus clearly overfitting,

but still performing well on the test set. The other techniques provide slightly improved performance on the training data, as expected.

Technique/Accuracy (%)	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	96.2 ± 0.1	95.7	96.8	95.5	97.0
Naive Bayes	81.2 ± 0.0	81.2	81.2	81.2	81.2
Logistic Regression	78.8 ± 0.0	78.8	78.8	78.8	78.8
Multilayer Perceptron Network	92.6 ± 0.1	91.5	93.6	90.7	94.2
SVM	85.2 ± 0.0	85.2	85.2	85.2	85.2
Random Forest	99.1 ± 0.0	98.8	99.4	98.6	99.7

Table 12: The *Axis & Allies* Dataset – accuracy on training data.
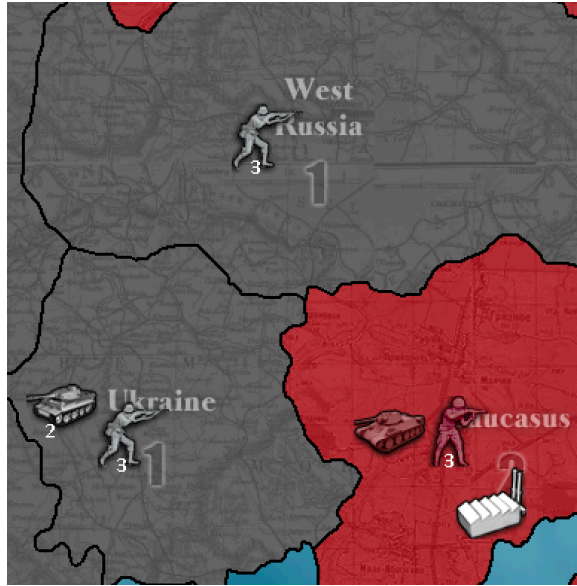
Figure 8: The *Axis & Allies* mini game.

5.4 The Noisy XOR Dataset with Non-informative Features

We now turn to an artifical dataset, constructed to uncover "blind zones" caused by XOR-like relations. Furthermore, the dataset contains a large number of random non-informative features to measure susceptibility towards the curse of dimensionality [37]. To examine robustness towards noise, we have further randomly inverted 40% of the outputs.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	y
0	1	0	1	1	0	1	1	0	1	1	0	1
1	1	1	0	1	0	1	1	0	0	1	1	0
0	0	1	1	0	1	1	1	1	0	1	0	0
1	1	1	0	1	1	1	0	1	1	0	0	1

Table 13: The Noisy XOR Dataset with Non-informative Features.

The dataset consists of 10 000 examples with twelve binary inputs, $X = [x_1, x_2, \ldots, x_{12}]$, and a binary output, y. Ten of the inputs are completely random. The two remaining inputs, however, are related to the output y through an XOR-relation, $y = \text{XOR}(x_{k_1}, x_{k_2})$. Finally, 40% of the outputs are inverted. Table 13 shows four examples from the dataset, demonstrating the high level of noise. We partition the dataset into training and test data, using 50% of the data for training.

The Tsetlin Machine[7] used here contains 20 clauses, and uses an s-value of 3.9 and a summation target T of 15. Furthermore, the individual Tsetlin Automata each has 100 states. The Tsetlin Machine is run for 200 epochs, and it is the accuracy after the final epoch, that we report.

Table 14 contains four of the clauses produced by the Tsetlin Machine. Notice how the noisy dataset from Table 13 has been turned into informative propositional formulas that capture the structure of the dataset.

The empirical results are found in Table 15. Again, we report mean accuracy with 95% confidence intervals, 5 and 95 percentiles, as well as the minimum and maximum accuracy obtained, across the 100 replications of the experiment. Note that for the test data, the output values are unperturbed. As seen, the XOR-relation, as expected, makes Logistic Regression and the Naive Bayes Classifier incapable of predicting the output value y, resorting to random guessing. Both the neural network and the Tsetlin Machine, on the other hand, see through the noise and captures the underlying XOR pattern. SVM performs slightly better than the Naive Bayes Classifier and Logistic Regression, however, is clearly distracted by the added non-informative features (the SVM performs much better with fewer non-informative features).

Figure 9 shows how accuracy degrades with less data, when we vary the dataset size from 1000 examples to 20 000 examples. As expected, Naive Bayes and Logistic Regression guess blindly for all the different data sizes. The main observation, however, is that the accuracy advantage the Tsetlin Machine has over neural networks increases with less training data. Indeed, it turns out that the Tsetlin Machine performs robustly with small training data sets in all of our experiments.

5.5 The MNIST Dataset

We next evaluate the Tsetlin Machine on the MNIST Dataset of Handwritten Digits[8] [38], also investigating how learning progresses, epoch-by-epoch, in terms of accuracy. Note that the experimental results reported here can be reproduced with the demo found at
https://github.com/cair/fast-tsetlin-machine-with-mnist-demo.

The original dataset consists of 60 000 training examples, and 10 000 test examples. We binarize this dataset by replacing pixel values larger than 0.3 with 1. Pixel values below or equal to 0.3 are replaced with 0.

The Tsetlin Machine[9] employed here contains 40 000 clauses, 4000 clauses per class, uses an

---

[7]In this experiment, we used a Multi-Class Tsetlin Machine, described in Section 6.1.

[8]http://www.pymvpa.org/datadb/mnist.html

[9]In this experiment, we used a Multi-Class Tsetlin Machine, described in Section 6.1. We also applied

No.	Sign	Clause Learned
1	+	$\neg x_1 \wedge x_2$
2	−	$\neg x_1 \wedge \neg x_2$
3	+	$x_1 \wedge \neg x_2$
4	−	$x_1 \wedge x_2$

Table 14: Example of four clauses composed by the Tsetlin Machine for the XOR Dataset with Non-informative Features.

Technique/Accuracy (%)	Mean	5 %ile	95 %ile	Min.	Max.
Tsetlin Machine	99.3 ± 0.3	95.9	100.0	91.6	100.0
Naive Bayes	49.8 ± 0.2	48.3	51.0	41.3	52.7
Logistic Regression	49.8 ± 0.3	47.8	51.1	41.1	53.1
Multilayer Perceptron Network	95.4 ± 0.5	90.1	98.6	88.2	99.9
SVM	58.0 ± 0.3	56.4	59.2	55.4	66.5

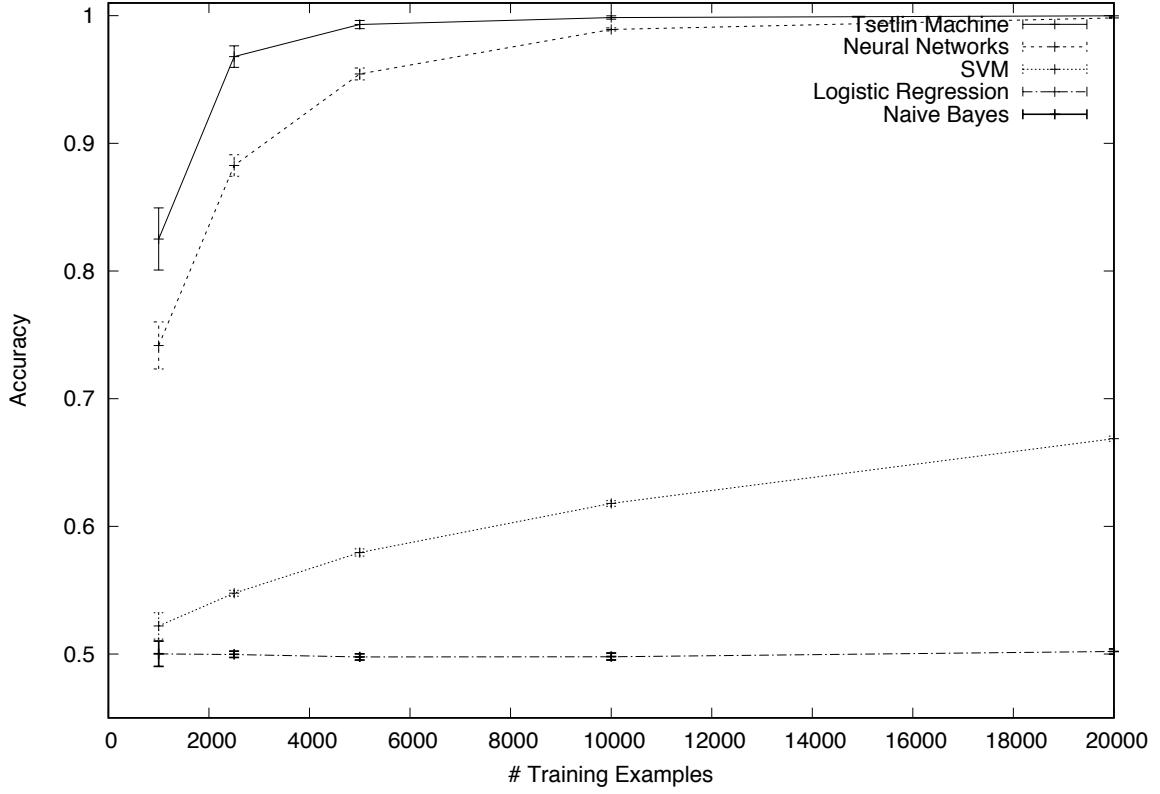Table 15: The Noisy XOR Dataset with Non-informative Features – accuracy on test data.

Figure 9: Accuracy (y-axis) for the Noisy XOR Dataset for different training dataset sizes (x-axis).
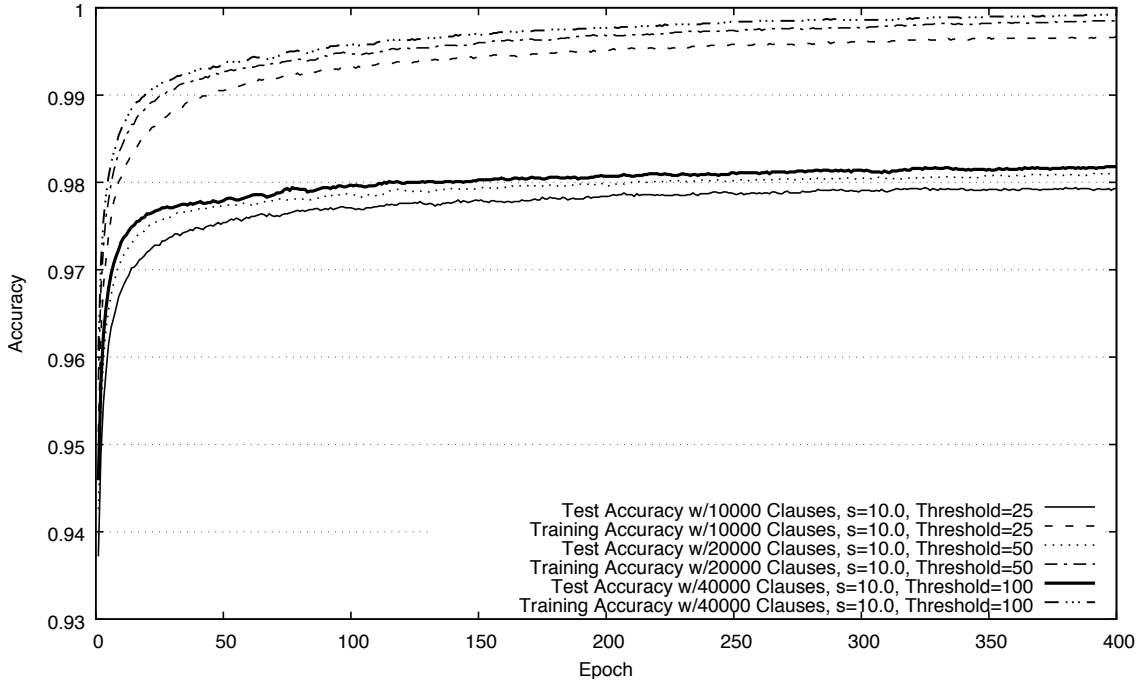

Figure 10: The mean test- and training accuracy per epoch for the Tsetlin Machine on the MNIST Dataset.

s-value of 10.0, and a summation target T of 50. Furthermore, the individual Tsetlin Automata each has 256 states. The Tsetlin Machine is run for 400 epochs, and it is the accuracy after the final epoch that is reported.

Boosting of True Positive Feedback to *Include* actions, as described in Section 3.3.3.

As seen in Figure 10, both mean test- and training accuracy increase almost monotonically across the epochs, however, affected by random fluctuation. Perhaps most notably, while the mean accuracy on the training data approaches 99.9%, accuracy on the test data continues to increase as well, hitting 98.2% after 400 epochs. This is quite different from what occurs with back-propagation on a neural network, where accuracy on test data starts to drop at some point due to overfitting, without proper regularization mechanisms.

The figure also shows how varying the number of clauses and the summation target T affects accuracy and learning stability. With more clauses available to express patterns, in combination with a higher summation target T, both learning speed, stability and accuracy increases, however, at the expense of larger computational cost.

Technique	Accuracy (%)
2-layer NN, 800 HU, Cross-Entropy Loss	*98.6*
Tsetlin Machine (95 %ile)	98.3
Tsetlin Machine (Mean)	98.2 ± 0.0
Tsetlin Machine (5 %ile)	98.1
K-nearest-neighbors, L3	*97.2*
3-layer NN, 500+150 hidden units	*97.1*
40 PCA + quadratic classifier	*96.7*
1000 RBF + linear classifier	*96.4*
Logistic regression	91.5
Linear classifier (1-layer NN)	*88.0*
Decision tree	87.8
Multinomial Naive Bayes	83.2

Table 16: A comparison of vanilla machine learning algorithms with the Tsetlin Machine, directly on the original unenhanced MNIST dataset (NN - Neural Network).

Table 16 reports the mean accuracy of the Tsetlin Machine, across the 50 experiment runs we executed. As points of reference, results for other well-known algorithms have been obtained from http://yann.lecun.com/exdb/mnist/ and included in the table (in italic). Only the vanilla version of the algorithms, that has been applied directly on unenhanced MNIST data, is included here. The purpose of this selection is to strictly compare algorithmic performance. In other words, we do not consider the effect of enhancing the dataset (e.g., warping, distortion, deskewing), combining different algorithms (e.g., neural network combined with nearest neighbor, convolution schemes), or applying meta optimization techniques (boosting, ensemble learning, etc.). With such techniques, it is possible to significantly increase accuracy, with the best currently reported results being an accuracy of 99.79% [39]. Enhancing the vanilla Tsetlin Machine with such techniques is further work.

Additionally, as a further point of reference, we train and evaluate logistic regression, decision trees, and multinomial Naive Bayes on the *binarized* MNIST dataset used by the Tsetlin Machine.

As seen in the table, the Tsetlin Machine provides competitive accuracy, outperforming e.g. K-nearest neighbor and a 3-layer neural network. It is outperformed by a 2-layer neural network with 800 hidden nodes, using cross entropy loss. However, note that the Tsetlin Machine operates upon the binarized MNIST data (the grey tone value of each pixel is either set to 0 or 1), and thus has a disadvantage. Improved binarization techniques for the Tsetlin Machine is further work.

6 The Tsetlin Machine as a Building Block in More Advanced Architectures

We have designed the Tsetlin Machine to facilitate building of more advanced architectures. We will here exemplify different ways of connecting multiple Tsetlin Machines in more advanced

architectures.

6.1 The Multi-Class Tsetlin Machine

In some pattern recognition problems the task is to assign one of n classes to each observed pattern X. That is, one needs to decide upon a *single* output value, $y \in \{1, \ldots, m\}$. Such a multi-class pattern recognition problem can be handled by the Tsetlin Machine by representing y as bits, using multiple outputs y. In this section, however, we present an alternative architecture that addresses the multi-class pattern recognition problem more directly.
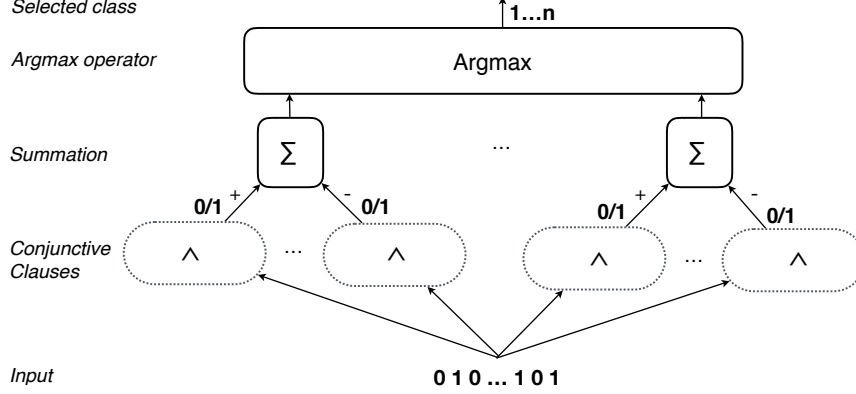
Figure 11: The Multi-Class Tsetlin Machine.

Figure 11 depicts the Multi-Class Tsetlin Machine[10] which replaces the threshold function of each output $y, i \in \{1, \ldots, m\}$, with a single **argmax** operator. With the **argmax** operator, the index i of the largest sum $\sum_{j=1}^{n/2} C_j^{1,i}(X) - \sum_{j=1}^{n/2} C_j^{0,i}(X)$ is outputted as the final output of the Tsetlin Machine:

$$y = \operatorname{argmax}_{i=1,\ldots,m} \left(\sum_{j=1}^{n/2} C_j^{1,i}(X) - \sum_{j=1}^{n/2} C_j^{0,i}(X) \right). \tag{63}$$

Training is done as described in Section 3, apart from one critical modification. Assume we have $y = i$ for the current observation (X, y). Then the Tsetlin Automata teams associated with class i are trained as per $y = 1$ in the original Algorithm 1. Additionally, a random class $q \neq i$ is selected. The Tsetlin Automata teams of class q are then trained in accordance with $y = 0$ in the original algorithm.

6.2 The Fully Connected Deep Tsetlin Machine

Another architectural family is the Fully Connected Deep Tsetlin Machine [40], illustrated in Figure 12. The purpose of this architecture is to build composite propositional formulas, combining the propositional formula composed at one layer into more complex formula at the next. As exemplified in the figure, we here connect multiple Tsetlin Machines in a sequence. The clause output from each Tsetlin Machine in the sequence is provided as input to the next Tsetlin Machine in the sequence. In this manner we build a multi-layered system. For instance, if layer t produces two clauses $(P \wedge \neg Q)$ and $(\neg P \wedge Q)$, layer $t+1$ can manipulate these further, treating them as inputs. Layer $t+1$ could then form more complex formulas like $\neg(P \wedge \neg Q) \wedge (P \wedge \neg Q)$, which can be rewritten as $(\neg P \vee Q) \wedge (P \wedge \neg Q)$.

One simple approach for training such an architecture is indicated in the figure. As illustrated, each layer is trained independently, directly from the output target y, exactly as described in Section 3.3. The training procedure is thus similar to the strategy Hinton et al.

---

[10]An implementation of the Multi-Class Tsetlin Machine can be found at https://github.com/cair/TsetlinMachine.
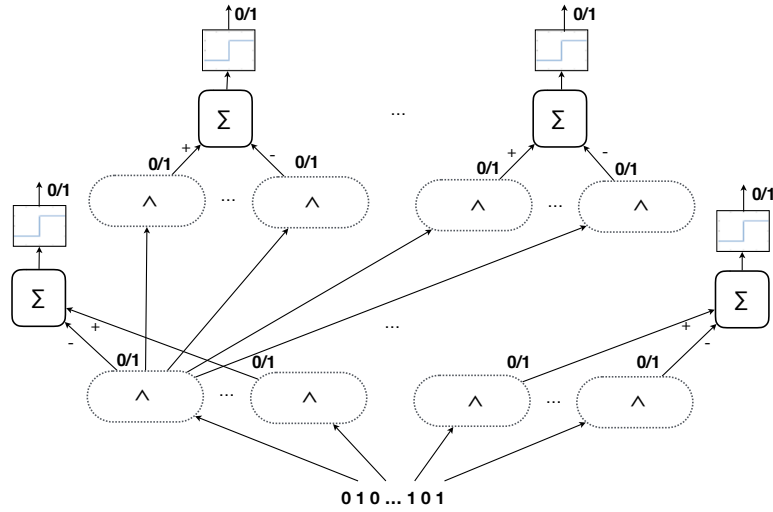
Figure 12: The fully connected Deep Tsetlin Machine.

used to train their pioneering Deep Belief Networks, layer-by-layer, in 2006 [41]. Such an approach can be effective when each layer produces abstractions, in the form of clauses, that can be taken advantage of in the following layer.

6.3 The Convolutional Tsetlin Machine

We next demonstrate how self-contained and independent Tsetlin Machines can interact to build a Convolutional Tsetlin Machine [42], illustrated in Figure 13. The Convolutional Tsetlin
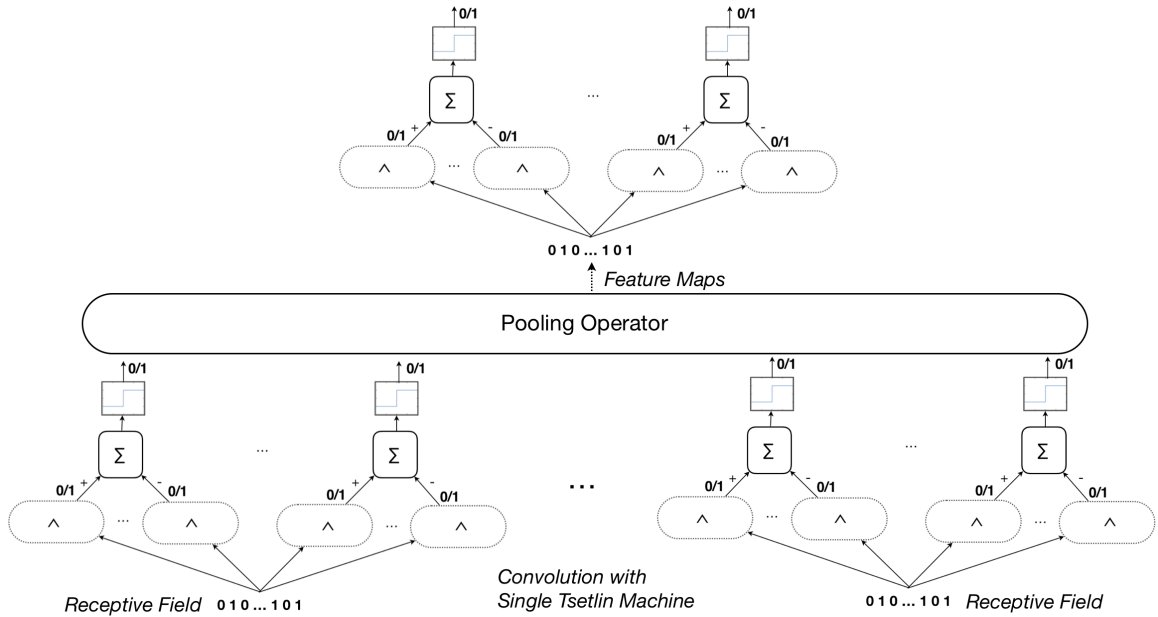
Figure 13: The Convolutional Tsetlin Machine.

Machine is a deep architecture based on mathematical convolution, akin to Convolutional Neural Networks [38]. For illustration purposes, consider 2D images of size 100×100 as input. At the core of a Convolutional Tsetlin Machine we find a kernel Tsetlin Machine with a small receptive field. Each layer t of the Convolutional Tsetlin Machine operates as follows:

1. A convolution is performed over the input from the previous Tsetlin Machine layer, producing one *feature map* per output y. Here, the Tsetlin Machine acts as a kernel

in the convolution. In this manner, we reduce complexity by reusing the same Tsetlin Machine across the whole image, focusing on a small image patch at a time.

2. The feature maps produced are then down-sampled using a pooling operator, in a similar fashion as done in a Convolutional Neural Network, before the next layer and a new Tsetlin Machine takes over. Here, the purpose of the pooling operation is to gradually increase the abstraction level of the clauses, layer by layer.

A simple approach for training a Convolutional Tsetlin Machine is indicated in the figure. In brief, the feedback to the Tsetlin Machine kernel is directly provided from the desired end output y, exactly as described in Section 3. The only difference is the fact that the input to layer $t + 1$ comes from the down-scaled feature map produced by layer t. Again, this is useful when each layer produces abstractions, in the form of clauses, that can be taken advantage of at the next layer.

6.4 The Recurrent Tsetlin Machine

The final example is the Recurrent Tsetlin Machine [43] (Figure 14). In all brevity, the same Tsetlin Machine is here reused from time step to time step. By taking the output from the Tsetlin Machine of the previous time step as input, together with an external input from the current time step, an infinitely deep sequence of Tsetlin Machines is formed. This is quite similar to the family of Recurrent Neural Networks [44]. Again, the architecture can be trained
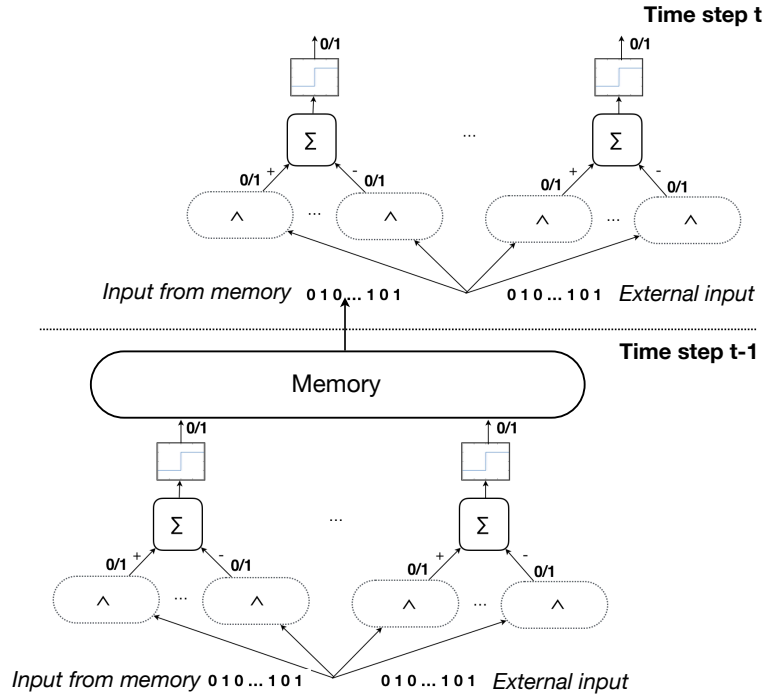

Figure 14: The Recurrent Tsetlin Machine.

layer by layer, directly from the target output $y(t)$ of the current time step t. However, to learn more advanced sequential patterns, there is a need for rewarding and penalizing that propagate back in time. How to design such a propagation scheme is presently an open research question.

7 Conclusion and Further Work

In this paper we proposed the Tsetlin Machine, an alternative to neural networks. The Tsetlin Machine solves the vanishing signal-to-noise ratio of collectives of Tsetlin Automata. This allows it to coordinate millions of Tsetlin Automata. By equipping teams of Tsetlin Automata

with the ability to express patterns in propositional logic, we have enabled them to recognize complex patterns. Furthermore, we proposed a novel decentralized feedback orchestration mechanism. The mechanism is based on resource allocation principles, with the intent of maximizing effectiveness of sparse pattern recognition capacity. This mechanism effectively provides the Tsetlin Machine with the ability to capture unlabelled sub-patterns.

Our theoretical analysis reveals that the Tsetlin Machine game have Nash equilibria that maps to propositional formulas that maximize pattern recognition accuracy. In other words, there are no local optima in the learning process, only global ones. This explains how the collectives of Tsetlin Automata are able to accurately converge towards complex propositional formulas that capture the essence of five diverse pattern recognition problems.

Overall, the Tsetlin Machine seems to be particularly suited for digital computers, being based on simple bit manipulation with AND-, OR-, and NOT gates. Both input, hidden patterns, and output can be expressed as bit patterns.

In our empirical evaluations on five benchmarks, the Tsetlin Machine provided competitive accuracy with respect to both Multilayer Perceptron Networks, Support Vector Machines, Decision Trees, Random Forests, the Naive Bayes Classifier and Logistic Regression. It further turns out that the Tsetlin Machine requires less data than neural networks, outperforming the Naive Bayes Classifier in data sparse environments.

By demonstrating that the longstanding problem of vanishing signal-to-noise ratio can be solved, the Tsetlin Machine further provides a novel game theoretic framework for recasting the problem of pattern recognition. This framework can thus provide opportunities for introducing bandit algorithms into large-scale pattern recognition. It could for instance be interesting to investigate the effect of replacing the Tsetlin Automaton with alternative bandit algorithms, such as algorithms based on Thompson Sampling [45, 46, 47, 48] or Upper Confidence Bounds [49].

The more advanced Fully Connected Deep Tsetlin Machine, the Convolution Tsetlin Machine, and the Recurrent Tsetlin Machine architectures also form a starting point for further exploration. These architectures can potentially improve pattern representation compactness and learning speed. However, it is currently unclear how these architectures can be most effectively trained.

Lastly, the high accuracy of the Tsetlin Machine, combined with its ability to produce self-contained easy-to-interpret propositional formulas for pattern recognition, makes it attractive for applied research, such as in the safety-critical medical domain.

Acknowledgements

Code Availability

Source code and datasets for the Tsetlin Machine, available under the MIT Licence, can be found at https://github.com/cair/TsetlinMachine and https://github.com/cair/pyTsetlinMachine.

Data Availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," pp. 436–444, 2015.

[2] M. L. Tsetlin, "On behaviour of finite automata in random medium," *Avtomat. i Telemekh*, vol. 22, no. 10, pp. 1345–1354, 1961.

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

[4] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, 1952.

[5] J. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979.

[6] J. Carroll, *Theory of Finite Automata With an Introduction to Formal Languages.* Prentice Hall, 1989.

[7] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 3, pp. 360–375, 1985.

[8] P. Sastry and M. Thathachar, "Learning automata algorithms for pattern classification," *Sadhana*, vol. 24, no. 4-5, pp. 261–292, 1999.

[9] S.-H. Zahiri, "Learning automata based classifier," *Pattern Recognition Letters*, vol. 29, no. 1, pp. 40–48, 2008.

[10] ——, "Classification rule discovery using learning automata," *International Journal of Machine Learning and Cybernetics*, vol. 3, no. 3, pp. 205–213, 2012.

[11] P. Sastry, G. Nagendra, and N. Manwani, "A team of continuous-action learning automata for noise-tolerant learning of half-spaces," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 1, pp. 19–28, 2009.

[12] M. Goodwin, A. Yazidi, and T. M. Jonassen, "Distributed learning automata for solving a classification task," in *2016 IEEE congress on evolutionary computation (CEC).* IEEE, 2016, pp. 3999–4006.

[13] H. Motieghader, A. Najafi, B. Sadeghi, and A. Masoudi-Nejad, "A hybrid gene selection algorithm for microarray cancer classification using genetic algorithm and learning automata," *Informatics in Medicine Unlocked*, vol. 9, pp. 246–254, 2017.

[14] M. A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Kluwer Academic Publishers, 2004.

[15] B. Tung and L. Kleinrock, "Using Finite State Automata to Produce Self-Optimization and Self-Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 4, pp. 47–61, 1996.

[16] 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*, vol. 27, no. 4, pp. 733–739, 1997.

[17] B. J. Oommen and D. C. Ma, "Deterministic Learning Automata Solutions to The Equipartitioning Problem," *IEEE Transactions on Computers*, vol. 37, no. 1, pp. 2–13, 1988.

[18] M. Ghavipour and M. R. Meybodi, "A streaming sampling algorithm for social activity networks using fixed structure learning automata," *Applied Intelligence*, 2018.

[19] A. Yazidi and B. John Oommen, "On the analysis of a random walk-jump chain with tree-based transitions and its applications to faulty dichotomous search," *Sequential Analysis*, vol. 37, pp. 31–46, jan 2018.

[20] J. Zhang, Y. Wang, C. Wang, and M. Zhou, "Symmetrical Hierarchical Stochastic Searching on the Line in Informative and Deceptive Environments," *IEEE Transactions on Cybernetics*, vol. 47, no. 3, pp. 626 – 635, jul 2016.

[21] B. J. Oommen, S. Misra, and O.-C. Granmo, "Routing bandwidth-guaranteed paths in MPLS traffic engineering: a multiple race track learning approach," *Computers, IEEE Transactions on*, vol. 56, no. 7, pp. 959–976, 2007.

[22] O.-C. Granmo and S. Glimsdal, "Accelerated Bayesian learning for decentralized two-armed bandit based decision making with applications to the Goore Game," *Applied Intelligence*, vol. 38, no. 4, 2013.

[23] J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 1947.

[24] O.-C. Granmo, B. J. Oommen, S. A. Myrer, and M. G. Olsen, "Learning Automata-based Solutions to the Nonlinear Fractional Knapsack Problem withApplications to Optimal Resource Allocation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 37, no. 1, pp. 166–175, 2007.

[25] V. Haugland, M. Kjølleberg, S.-E. Larsen, and O.-C. Granmo, "A two-armed bandit collective for hierarchical examplar based mining of frequent itemsets with applications to intrusion detection," *Transactions on Computational Collective Intelligence XIV*, vol. 8615, pp. 1–19, 2014.

[26] Guolong Su, Kush R. Varshney, and Dmitry M. Malioutov, "Interpretable Two-Level Boolean Rule Learning for Classification," in *ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*, 2016, pp. 66–70.

[27] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Jones, E. Klampfl, P. Macneille, and M. Gupta, "A Bayesian Framework for Learning Rule Sets for Interpretable Classification," *Journal of Machine Learning Research*, 2017.

[28] L. G. Valiant, "A Theory of the Learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[29] O.-C. Granmo, B. J. Oommen, S. A. Myrer, and M. G. Olsen, "Learning Automata-based Solutions to the Nonlinear Fractional Knapsack Problem with Applications to Optimal Resource Allocation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 37, no. 1, pp. 166–175, 2007.

[30] D. Monderer and L. Shapley, "Potential games," *Games and Economic Behavior*, vol. 14, no. 1, pp. 124–143, 1996. [Online]. Available: https://EconPapers.repec.org/RePEc:eee:gamebe:v:14:y:1996:i:1:p:124-143

[31] V. U. Krylov and M. Tsetlin, "Games Between Automata," *Automation and Remote Control*, vol. 24, pp. 889–899, 1963.

[32] V. A. Borovikov and V. I. Bryzgalov, "The Simplest Symmetric Game of Many Automata," *Avtomatika i Telemekhanika*, vol. 26, pp. 637–687, 1965.

[33] V. A. Volkonskii, "Asymptotic Properties of the Behavior of Elementary Automata in a Game," *Problemy Peredachi Informatsii*, vol. 1, pp. 36–53, 1965.

[34] B. G. Pittel, "The Asymptotic Properties of One Form of Goore Game," *Problemy Peredachi Informatsii*, vol. 1, pp. 99–112, 1965.

[35] R. Viswanathan and K. S. Narendra, "Games of stochastic automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-4, no. 1, pp. 131–135, 1974.

[36] K. S. Narendra, "Competitive and cooperative games of variable-structure stochastic automata," *Journal of Cybernetics*, vol. 3, no. 1, pp. 1–23, 1973.

[37] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. New York, NY: John Wiley and Sons, Inc., 2000.

[38] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278 – 2324, 1998.

[39] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," *International Conference on Machine Learning (ICML)*, 2013.

[40] O.-C. Granmo, "The Fully Connected Deep Tsetlin Machine," *In Preparation*, 2019.

[41] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527 – 1554, 2006.

[42] O.-C. Granmo, "The Convolutional Tsetlin Machine," *In Preparation*, 2019.

[43] ——, "The Recurrent Tsetlin Machine," *In Preparation*, 2019.

[44] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[45] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[46] O.-C. Granmo, "Solving Two-Armed Bernoulli Bandit Problems Using a Bayesian Learning Automaton," *International Journal of Intelligent Computing and Cybernetics*, vol. 3, no. 2, pp. 207–234, 2010.

[47] B. C. May, N. Korda, A. Lee, and D. S. Leslie, "Optimistic Bayesian sampling in contextual-bandit problems," *Journal of Machine Learning Research*, vol. 13, pp. 2069–2106, 2012.

[48] O. Chapelle and L. Li, "An Empirical Evaluation of Thompson Sampling," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2249–2257.

[49] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," in *Journal of Machine Learning Research*, vol. 3, no. 3, 2003, pp. 397–422.