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. Unknown to many, there exists an arguably even simpler and more versatile learning mechanism, namely, the Tsetlin Automaton. 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 easy-to-interpret *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*. Our theoretical analysis establishes that the Nash equilibria of the game are aligned with the propositional formulas that provide optimal pattern recognition accuracy. This translates to learning without local optima, only global ones. We argue that the Tsetlin Machine finds the propositional formula that provides optimal accuracy, with probability arbitrarily close to unity. In four distinct benchmarks, the Tsetlin Machine outperforms both Neural Networks, SVMs, Random Forests, the Naive Bayes Classifier and Logistic Regression. It further turns out that the accuracy advantage of the Tsetlin Machine increases with lack of data. The Tsetlin Machine has a significant computational performance advantage since both inputs, patterns, and outputs are expressed as bits, while recognition of patterns relies on bit manipulation. The combination of accuracy, interpretability, and computational simplicity makes the Tsetlin Machine a promising tool for a wide range of domains, including safety-critical medicine. Being the first of its kind, we believe the Tsetlin Machine will kick-start completely new paths of research, with a potentially significant impact on the AI field and the applications of AI.

1 Introduction

Although simple individually, artificial neurons provide state-of-the-art performance when interconnected in deep networks [1]. Highly successful, 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 late 1950s [2].

In this paper, we introduce the Tsetlin Machine — the first learning machine that solves complex pattern recognition problems with easy-to-interpret propositional formulas, composed by a collective of Tsetlin Automata. Using propositional logic to map an arbitrary sequence of input bits to an arbitrary sequence of output bits, the Tsetlin Machine is capable of outperforming classical machine learning approaches such as the Naive Bayes Classifier, Support Vector Machines, Logistic Regression, Random Forests, and even neural networks.

*Source code and datasets for the Tsetlin Machine can be found at `https://github.com/cair/TsetlinMachine`.

†Author's status: *Professor* and *Director*. This author can be contacted at: Centre for Artificial Intelligence Research (CAIR), University of Agder, Grimstad, Norway. E-mail: `ole.granmo@uia.no`

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.

The Tsetlin Automaton is one of the pioneering solutions to the well-known multi-armed bandit problem. It performs actions sequentially in an environment, and each action triggers either a reward or a penalty. The probability that an action triggers a reward is unknown to the automaton and may even change over time. Under such challenging conditions, the goal is to identify the action with the highest reward probability using as few trials as possible.

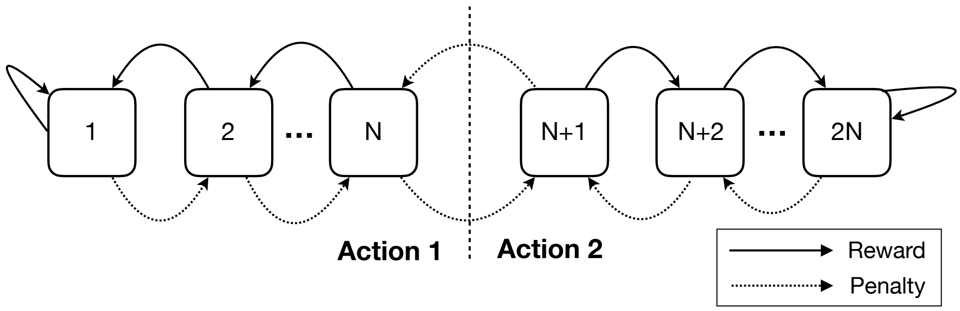

Figure 1: A Tsetlin Automaton for two-action environments.

The mechanism driving a Tsetlin Automaton is surprisingly simple. Figure 1 depicts a two-action Tsetlin Automaton published by M. L. Tsetlin in 1961 [2]. As illustrated, a Tsetlin Automaton is simply a fixed finite-state automaton [4] 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 is performed in the states with index 1 to N, while Action 2 is performed in 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), and one set of state transitions is activated on penalty (dotted lines). As seen, rewards and penalties trigger specific transitions from one state to another, designed to reinforce successful actions (those eliciting rewards).

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 in the figure. *The Tsetlin Automaton is thus extremely simple computationally, with a very small memory footprint.*

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

Amazingly, the simple Tsetlin Automaton approach has formed the core for more advanced finite state learning automata designs that solve a wide class of problems. This includes resource allocation [5], decentralized control [6], knapsack problems [7], searching on the line [8, 9], meta-learning [10], the satisfiability problem [11, 12], graph colouring [13], preference learning [14], frequent itemset mining [15], adaptive sampling [16], spatio-temporal event detection [17], equi-partitioning [18], learning in deceptive environments [19], as well as routing in telecommunication networks [20]. The unique strength of all of these finite state learning automata solutions is that they provide 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]

[1]Note that there exists another family of learning automata, referred to as *variable structure learning automata* [21]. Although still simple, these are significantly more complex than the Tsetlin Automaton because they need

The ability to handle stochastic and unknown environments for a surprisingly wide range of problems, combined with its computational simplicity and small memory footprint, make the Tsetlin Automaton an attractive building block for complex machine learning tasks. However, the success of the Tsetlin Automaton has been hindered by a particularly adverse challenge, explored by Kleinrock and Tung in 1996 [6]. Complex problem solving requires teams of interacting Tsetlin Automata, and unfortunately, each team member introduces noise. This is due to the inherently decentralized and stochastic nature of Tsetlin Automata based decision-making. 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 Tsetlin Automata grows, the level of noise increases. We refer to this effect as the *vanishing signal-to-noise ratio problem*. This vanishing signal-to-noise-ratio demands in the end an infinite number of states per Tsetlin Automaton, which in turn, leads to impractically slow convergence [3, 6].

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., a neural network. It is also efficient to evaluate for computers. Despite this simplicity, the Tsetlin Machine produces bit patterns that deliver state-of-the-art pattern recognition accuracy (see Section 5).

1.3 Paper Contributions

To attack the challenge of limited pattern expression capability and vanishing signal-to-noise ratio, in this paper we introduce the *Tsetlin Machine*. The contributions of the paper can be summarized as follows:

- The Tsetlin Machine 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. The game we have designed allows thousands of Tsetlin Automata to successfully cooperate.
- Our theoretical analysis establishes that the Nash equilibria of the game are aligned with the propositional formulas that provide optimal pattern recognition accuracy. This translates to learning without local optima, only global ones.
- We further argue formally that the Tsetlin Machine finds the propositional formula that provides optimal pattern recognition accuracy, with probability arbitrarily close to unity.
- The game orchestrated by the Tsetlin Machine is based on resource allocation principles [7]. By allocating sparse pattern representation resources, the Tsetlin Machine is able

to maintain an action probability vector for sampling actions. Although promising, their success in pattern recognition has been limited to small scale problems, being restricted by constrained pattern representation capability (linearly separable classes and simple decision trees) [21, 22, 3] and the problem of vanishing signal-to-noise-ratio [6], solved in the present paper.

to capture intricate unlabelled sub-patterns, for instance addressing the renowned Noisy XOR-problem.

- 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 a simple 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 particularly suited for digital computers, being merely based on bit manipulation with AND-, OR-, and NOT operators. Both input, hidden patterns, and output are represented as bit patterns.
- In thorough empirical evaluations on four diverse datasets, the Tsetlin Machine outperforms both single layer neural networks, Support Vector Machines, Random Forests, the Naive Bayes Classifier, and Logistic Regression.
- It further turns out that the Tsetlin Machine requires significantly less data than neural networks, outperforming even the Naive Bayes Classifier in data sparse environments.
- We demonstrate how the Tsetlin Machine can be used as a building block to create more advanced architectures, including the Multi-Class Tsetlin Machine, the Fully Connected Deep Tsetlin Machine, the Convolutional Tsetlin Machine, and the Recurrent Tsetlin Machine.

The combination of accuracy, interpretability, and computational simplicity makes the Tsetlin Machine a promising tool for a wide range of domains, including safety-critical medicine. Being the first of its kind, we believe the Tsetlin Machine will kick-start completely new paths of research, with a potentially significant impact on the field of AI and the applications of AI.

1.4 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 propositional logic based pattern representation framework, before we introduce the Tsetlin Automata teams that write *conjunctive clauses* in propositional logic. These Tsetlin Automata teams are then organized to recognize complex patterns. We conclude the section by presenting the *Tsetlin Machine game* that we use to coordinate thousands of Tsetlin Automata, eliminating the vanishing signal-to-noise ratio problem.

In Section 4 we analyse pertintent 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 four diverse data sets: Flower categorization, digit recognition, board game planning, and the Noisy XOR Problem with Non-informative Features.

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 ahead.

2 The Pattern Recognition Problem

We here define the pattern recognition problem to be solved by the Tsetlin Machine, starting with the input to the system. The input to the Tsetlin Machine is an observation vector of o propositional variables, $X = [x_1, x_2, \ldots, x_o]$ with $x_k \in \{\mathbf{0}, \mathbf{1}\}$, $1 \leq k \leq o$. From this input vector, the Tsetlin Machine is to produce an output vector $Y = [y^1, y^2, \ldots, y^n]$ of n propositional variables, $y^i \in \{\mathbf{0}, \mathbf{1}\}$, $1 \leq i \leq n$.[2]

We assume that for each output, $y^i, 1 \leq i \leq n$, an independent underlying stochastic processes of arbitrary complexity randomly produces either $y^i = \mathbf{0}$ or $y^i = \mathbf{1}$, from the input $X = [x_1, x_2, \ldots, x_o]$. To deal with the stochastic nature of these processes, we employ probability theory for reasoning and modelling.

In all brevity, uncertainty is fully captured by the probability $P(y^i = \mathbf{1}|X)$, that is, the probability that y^i takes the value $\mathbf{1}$ given the input X. Being binary, the probability that y^i takes the value $\mathbf{0}$ then follows: $P(y^i = \mathbf{0}|X) = 1 - P(y^i = \mathbf{1}|X)$.

Under these conditions, the optimal decision is to assign y^i the value $v \in \{\mathbf{0}, \mathbf{1}\}$ with the largest posterior probability[23]: $\text{argmax}_{v \in \{\mathbf{0},\mathbf{1}\}} P(y^i = v|X)$.

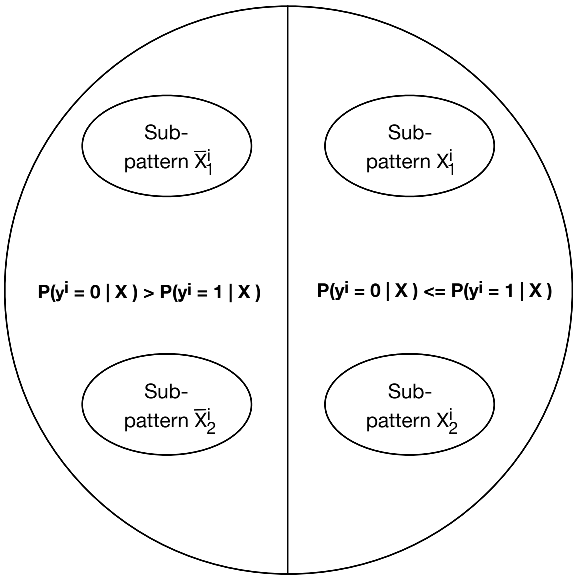

Figure 2: A partitioning of the input space according to the posterior probability of the output variable y^i, highlighting distinct sub-patterns in the input space, $X \in \mathcal{X}$. Sub-patterns most likely belonging to output $y^i = 1$ can be found on the right side, while sub-patterns most likely belonging to $y^i = 0$ on the left.

Now consider the complete set of possible inputs, $\mathcal{X} = \{x_1, \ldots, x_o \in [\mathbf{0}, \mathbf{1}]^o\}$. Each input X occurs with probability $P(X)$, and thus the joint input-output distribution becomes $P(X, y^i) = P(y^i|X)P(X)$.

As illustrated in Figure 2, the input space $\mathcal{X}$ can be partitioned in two parts, $\mathcal{X}^i = \{X | P(y^i = \mathbf{0}|X) \leq P(y^i = \mathbf{1}|X)\}$ and $\overline{\mathcal{X}}^i = \{X | P(y^i = \mathbf{0}|X) > P(y^i = \mathbf{1}|X)\}$. For observations in $\mathcal{X}^i$ it is optimal to assign y^i the value $\mathbf{1}$, while for partition $\overline{\mathcal{X}}^i$ it is optimal to assign y^i the value $\mathbf{0}$.

We now come to the crucial concept of unlabelled sub-patterns. As illustrated in the figure, $\mathcal{X}^i$ sub-divides further into s sub-parts, forming distinct sub-patterns, $\mathcal{X}^i_r, r = 1, \ldots, s$ (the same goes for $\overline{\mathcal{X}}^i$). Together, these sub-patterns span the whole input space, apart from a minimal level of "outlier" patterns that occur with probability, α, close to zero. That is, $P(\mathcal{X}^i_1 \cup \ldots \cup \mathcal{X}^i_s \cup \overline{\mathcal{X}}^i_1 \cup \ldots \cup \overline{\mathcal{X}}^i_s) = 1.0 - \alpha$, for a minimal α (the "outliers" remaining after a sub-division of the input space into s parts).

However, we only observe samples $(\hat{X}, \hat{y}^i)$ from the joint input-output distribution $P(X, y^i)$. Which sub-pattern $\hat{X}$ is sampled from is unavailable to us. We only know that each sub-pattern $\mathcal{X}^i_r$ occurs with probability $P(X^i_r) > \frac{1}{s}$ (by definition).

[2]Note that we have decided to use the binary representation $\mathbf{0}/\mathbf{1}$ to refer to the truth values *False*/*True*. These can be used interchangeably throughout the paper.

The challenging task we are going to solve is to learn all the sub-patterns merely by observing a limited sample from the joint input-output probability distribution $P(X, y^i)$, and by doing so, provide optimal pattern recognition accuracy.

3 The Tsetlin Machine

We now present the core concepts of the Tsetlin Machine in detail. We first present the propositional logic based pattern representation framework, before we introduce the Tsetlin Automata teams that write *conjunctive clauses* in propositional logic. These Tsetlin Automata teams are then organized to recognize complex sub-patterns. We conclude the section by presenting the Tsetlin Machine game that we use to coordinate thousands of Tsetlin Automata.

3.1 Expressing Patterns with Propositional Formulas

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 [24]. When critical patterns cannot be fully represented by the machine learning technique, accuracy suffers. Unfortunately, compared to the representation capability of the underlying language of digital computers, namely, Boolean algebra[3], most machine learning techniques appear rather limited, with neural networks being one of the exceptions. Indeed, with o input variables, $X = [x_1, x_2, \ldots, x_o]$, there are no less than 2^{2^o} unique propositional formulas, $f(X)$. Perhaps only a single one of them will provide optimal pattern recognition accuracy for the task at hand.

We equip the Tsetlin Machine with the full expression power of propositional logic. The representation power of propositional logic is perhaps best seen in light of the Satisfiabiliy (SAT) problem, which can be solved using a team of Tsetlin Automata [12]. The SAT problem is known to be NP-complete [25] and plays a central role in a number of applications in the fields of VLSI Computer-Aided design, Computing Theory, Theorem Proving, and Artificial Intelligence. Generally, a SAT problem is defined in so-called *conjunctive normal form.* To facilitate Tsetlin Automata based learning, we will instead represent patterns using *disjunctive normal form.*

Patterns in Disjunctive Normal Form. Briefly stated, we represent the relation between an input, $X = [x_1, x_2, \ldots, x_o]$, and the output, y^i, using a propositional formula Φ^i in disjunctive normal form:

$$\Phi^i = \bigvee_{j=1}^{m} C_j^i. \tag{1}$$

The formula consists of a disjunction of m conjunctive clauses, C_j^i. Each conjunctive clause in turn, represents a specific sub-pattern governing the output y^i.

Sub-Patterns and Conjunctive Clauses. Each clause C_j^i in the propositional formula Φ^i for y^i has the form

$$C_j^i = \left(\bigwedge_{k \in I_j^i} x_k \right) \wedge \left(\bigwedge_{k \in \bar{I}_j^i} \neg x_k \right). \tag{2}$$

That is, the clause is a conjunction of *literals*, where a literal is a propositional variable, x_k, without modification, or its negation $\neg x_k$. Here, I_j^i and $\bar{I}_j^i$ are non-overlapping subsets of the input variable indexes, $I_j^i, \bar{I}_j^i \subseteq \{1, ...o\}$, $I_j^i \cap \bar{I}_j^i = \emptyset$. The subsets decide which of the input variables take part in the clause, and whether they are negated or not. The input variables from I_j^i are included as is, while the input variables from $\bar{I}_j^i$ are negated.

For example, the propositional formula $(P \wedge \neg Q) \vee (\neg P \wedge Q)$ consists of two conjunctive clauses, and four literals, P, Q, $\neg P$, and $\neg Q$. The formula evaluates to **1** if

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

- $P = \mathbf{1}$ and $Q = \mathbf{0}$, or if
- $P = \mathbf{0}$ and $Q = \mathbf{1}$.

All other truth value assignments evaluate to $\mathbf{0}$, and thus the formula captures the renowned XOR-relation.

Definition 1 (The Problem of Pattern Learning with Propositional Logic)**.** *A set $\mathcal{S}$ of independent samples, $(\hat{X}, \hat{y}^i)$, from the joint input-output probability distribution $P(X, y^i)$ is provided. In the Problem of Pattern Learning with Propositional Logic, one must determine the propositional formula $\Phi^i(X)$ that evaluates to $\mathbf{0}$ iff $P(y^i = \mathbf{0}|X) > P(y^i = \mathbf{1}|X)$ and to $\mathbf{1}$ iff $P(y^i = \mathbf{0}|X) \leq P(y^i = \mathbf{1}|X)$, merely based on the samples in $\mathcal{S}$.*

The above problem decomposes into identifying the conjunctive clauses C^i_j whose disjunction evaluates to $\mathbf{1}$ iff $X \in \mathcal{X}^i$ (see Section 2 for the definition of $\mathcal{X}^i$).

3.2 The Tsetlin Automata Team for Composing Clauses

At the core of the Tsetlin Machine we find the conjunctive clauses, $C^i_j, j = 1, \ldots, m$, from Eqn. 2. For each clause C^i_j, we form a team of $2o$ Tsetlin Automata, two Tsetlin Automata per input variable x_k. Figure 3 captures the role of each of these Tsetlin Automata, and how they interact to form the clause C^i_j.

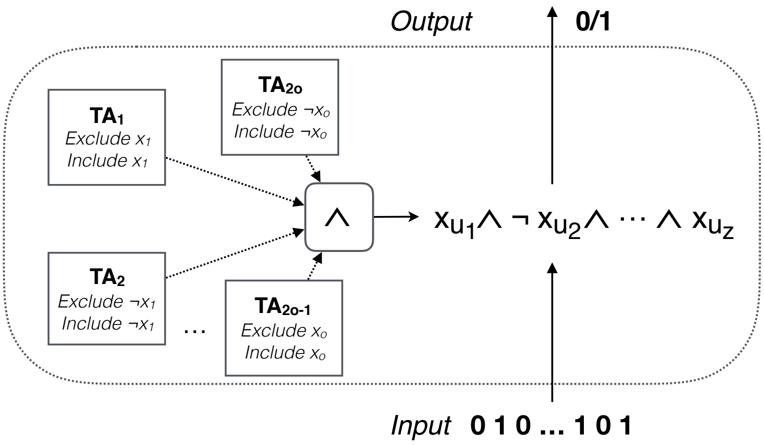

Figure 3: The Tsetlin Automata team for composing a clause.

As seen, o input variables, $X = [x_1, \ldots, x_o]$, are fed to the clause. The critical task of the Tsetlin Automata team is to decide which input variables to include in I^i_j and which input variables to include in $\bar{I}^i_j$. If a literal is excluded by its associated Tsetlin Automaton, it does not take part in the conjunction. As illustrated in the figure, Tsetlin Automaton TA_{2k-1} is responsible for deciding whether to "Include" or "Exclude" input variable x_k, while another Tsetlin Automaton, TA_{2k}, decides whether to "Include" or "Exclude" $\neg x_k$. That is, x_k can take part in the clause as is, take part in negated form, $\neg x_k$, or not take part at all.

As illustrated to the right in the figure, the clause is formed after each Tsetlin Automaton has made its decision, to include or exclude its associated literal. The current input $\hat{X}$ can then be transformed to an output, using the clause, $C^i_j(\hat{X}) = x_{u_1} \wedge \neg x_{u_2} \wedge \ldots \wedge x_{u_z}$, that has been formed by the automata team.

3.3 The Basic Tsetlin Machine Architecture

We are now ready to build the complete Tsetlin Machine. We do this by assigning m clauses, $C^i_j, j = 1, 2, \ldots, m$, to each output $y^i, i = 1, 2, \ldots, n$. The number of clauses m per output y^i is a meta-parameter that is decided by the number of sub-patterns associated with each y^i. If

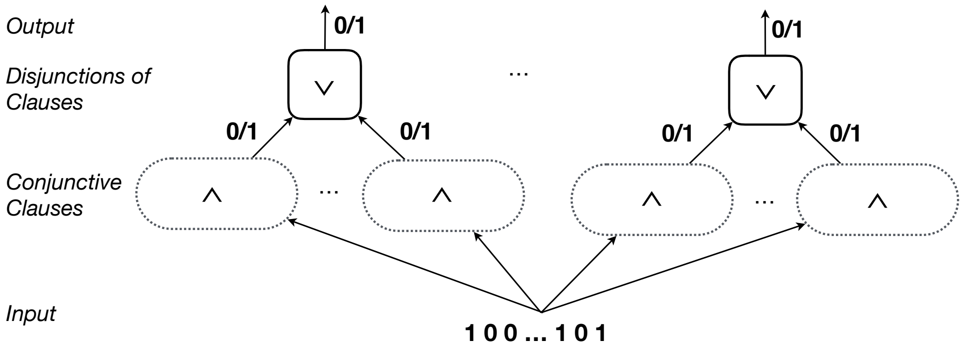

Figure 4: The basic Tsetlin Machine architecture.

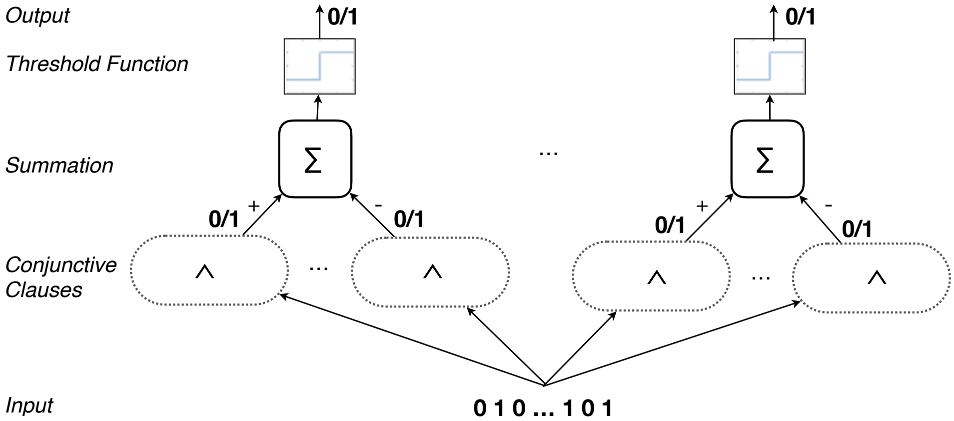

Figure 5: The extended Tsetlin Machine architecture, introducing clause polarity, a summation operator collecting "votes", and a threshold function arbitrating the final output.

the latter is unknown, an appropriate m can be found using a grid search, corresponding to selecting the number of hidden nodes in a neural network layer.

With the clause structure in place, we assign one Tsetlin Automata team, $\mathcal{G}_j^i = \{\mathrm{TA}_k^{ij} | 1 \leq k \leq 2o\}$, to each clause C_j^i. As shown in Figure 4, the architecture consists of $m \times n$ conjunctive clauses, each formed by an independent Tsetlin Automata team. Each Tsetlin Automata team thus governs the selection of which literals to include in their respective clauses, with the collective of teams jointly addressing the whole pattern recognition problem. As indicated, the clauses corresponds to the hidden layer of a neural network, although instead of having neurons with nonlinear activation functions, we have formulas in propositional logic that evaluates to **0** or **1**.

This architecture can thus express any formula in propositional logic, constrained by the number of clauses. Therefore, this basic architecture is interesting by itself. However, real-world problems do not necessarily fit the pattern recognition problem laid out in Section 2 exactly, thus raising the need for additional robustness. In the present paper, we therefore focus on a slightly more sophisticated version of the architecture, where the disjunction operator is replaced by a summation operator that accumulates votes, as well as a threshold function that decides the final output y^i. The added robustness helps solve the pattern recognition tasks presented in Section 5.

3.4 The Extended Tsetlin Machine Architecture

In order to render the architecture more robust towards noise, and to allow more compact representation of patterns, we now replace the OR-operator with a summation operator and a threshold function.

Figure 5 depicts the resulting extended architecture. Again, the architecture consists of a number of conjunctive clauses, each associated with a dedicated Tsetlin Automata team that decides which literals to include in each clause. However, instead of simply taking part in an OR relation, each clause, $C_j^i \in \mathcal{C}^i = \{C_j^i | j = 1, \ldots, m\}$, $i \in \{1, \ldots, n\}$, is now given a fixed polarity. For simplicity, we assign positive polarity to clauses with an odd index j, while clauses with an even index are assigned negative polarity. In the figure, polarity is indicated with a '+' or '-' sign, attached to the output of each clause.

Clauses, C_j^i, with positive polarity contribute to a final output of $y^i = \mathbf{1}$, while clauses with a negative polarity contribute towards a final output of $y^i = \mathbf{0}$. The contributions can be seen as votes, with each clause either casting a vote, $C_j^i(X) = \mathbf{1}$, or declining to vote, $C_j^i(X) = \mathbf{0}$. A positive vote means that the corresponding clause has recognized a sub-pattern associated with output $y^i = \mathbf{1}$, while a negative vote means that the corresponding clause has recognized a sub-pattern associated with the opposite output, $y^i = \mathbf{0}$.

After the clauses has produced their output, a summation operator, $\sum$, associated with the output y^i, sums the votes it receives from the clauses, $C_j^i, j = 1, \ldots, m$. Clauses with positive polarity contribute positively while those with negative polarity contribute negatively. Overall, the purpose is to reach a balanced output decision, weighting positive evidence against negative evidence (assuming m is even):

$$f_{\sum}(\mathcal{C}^i(X)) = \left(\sum_{j \in \{1,3,\ldots,m-1\}} C_j^i(X) \right) - \left(\sum_{j \in \{2,4,\ldots,m\}} C_j^i(X) \right) \tag{3}$$

The final output, y^i, is decided by a threshold function

$$f_t(x) = \begin{cases} \mathbf{0} & \text{for } x < 0 \\ \mathbf{1} & \text{for } x \geq 0 \end{cases}.$$

that outputs $\mathbf{1}$ if the outcome of the summation is larger than or equal to zero. Otherwise, it outputs $\mathbf{0}$. The final output y_j can thus be calculated directly from an input X simply by summing the signed output of the m conjunctive clauses, followed by activating the threshold function:

$$y^i = f_t(f_{\sum}(\mathcal{C}^i(X))). \tag{4}$$

As a result of the expression power of propositional logic, highly complex patterns can be captured by the Tsetlin Machine. Furthermore, it is arguably easier for humans to interpret propositional formula compared to a neural network. Finally, notice how all of the operators are highly suited for the digital circuits of modern computers.

The crucial remaining issue, then, is how to learn the conjunctive clauses from data, to obtain optimal pattern recognition accuracy. We attack this problem next.

3.5 The Tsetlin Machine Game for Learning Conjunctive Clauses

We here introduce a novel game theoretic learning mechanism that guides the Tsetlin Automata stochastically 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 large Tsetlin Automata teams that contain thousands of Tsetlin Automata.

The 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, a single Tsetlin Automaton has the power to completely disrupt a clause by introducing a contradiction. The payoffs of the game must therefore be designed carefully, so that the Tsetlin Automata are guided in the correct direction. To complicate further, an explicit enumeration of the payoffs is impractical due to the potentially tremendous size of the game matrix. Instead, the payoffs associated with each cell must be calculated on demand.

3.5.1 Design of the Payoff Matrix

We design the payoff matrix for the game based on the notion of:

- **True positive output.** We define *true positive output* as correctly providing output $y^i = \mathbf{1}$.
- **False negative output.** We define *false negative output* as incorrectly providing the output $y^i = \mathbf{0}$ when the output should have been $y^i = \mathbf{1}$.
- **False positive output.** We define *false positive output* as incorrectly providing the output $y^i = \mathbf{1}$ when the output should have been $y^i = \mathbf{0}$.

By progressively reducing false negative and false positive output, and reinforcing true positive output, we intend to guide the Tsetlin Automata towards optimal pattern recognition accuracy. This guiding is based on what we will refer to as Type I and Type II Feedback.

3.5.2 Type I Feedback – Combating False Negative Output

Type I Feedback is decided by two factors, connecting the players of the game together:

- The choices of the Tsetlin Automata team $\mathcal{G}_j^i$ as a whole, summarized by the output of the clause $C_j^i(X)$ (the truth value of the clause).
- The truth value of the literal $x_k/\neg x_k$ assigned to the Tsetlin Automaton $\mathrm{TA}^{ij}_{2k-1}/\mathrm{TA}^{ij}_{2k}$.

Table 2 contains the probabilities that we use to generate Type I Feedback. For instance, assume that:

1. Clause $C_j^i(X)$ evaluates to $\mathbf{1}$,
2. Literal x_k is $\mathbf{1}$, and
3. Automaton TA^{ij}_{2k-1} has selected the action *Include Literal*.

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

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 unaffected.

	Truth Value of Target Clause C_j^i	$\mathbf{1}$		$\mathbf{0}$	
	Truth Value of Target Literal $x_k/\neg x_k$	$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{0}$
Include Literal ($k \in I_j^i / k \in \bar{I}_j^i$)	$P(\text{Reward})$	$\frac{s-1}{s}$	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 ($k \notin I_j^i / k \notin \bar{I}_j^i$)	$P(\text{Reward})$	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 $x_k/\neg x_k$ in the clause C_j^i. Type I Feedback is triggered to increase the number of clauses that correctly evaluates to $\mathbf{1}$ for a given input X.

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 boosting rewarding of *Include Literal* 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 Literal* 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.

Brief analysis of the Type I Feedback. Notice how the reward probabilities are designed to "tighten" clauses up to a certain point. That is, the probability of receiving rewards when selecting *Include Literal* is larger than when selecting *Exclude Literal.* The ratio is controlled by the parameter s. In this manner, s effectively decides how "fine grained" patterns the clauses are going to capture. The larger the value of s, the more the Tsetlin Automata team is stimulated to include literals in the clause. The only countering force is the inputs, X, that do not match the clause, which obviously grows as s is increased (the clause is "tightened"). When these forces are in balance, we have a Nash equilibrium as discussed further in the next section.

The above mechanism is a critical part of the Tsetlin Machine, allowing learning of any sub-pattern X_j^i, no matter how infrequent, as long as $P(X_j^i)$ is larger than $\frac{1}{s}$. This novel mechanism is studied both theoretically and empirically in the two following sections. As a rule of thumb, a large s leads to more "fine grained" clauses, that is, clauses with more literals, while a small s produces "coarser" clauses, with less literals included.

3.5.3 Type II Feedback – Combating False Positive Output

Table 3 covers Type II Feedback, that is, feedback that combats false positive output. This type of feedback is triggered when the output is $y^i = \mathbf{1}$ when it should have been $y^i = \mathbf{0}$. Then we want to achieve the opposite of what we seek with Feedback Type I. In all brevity, we now seek to modify clauses that evaluate to **1**, so that they instead evaluate to **0**. To achieve this, for each offending clause, we identify the Tsetlin Automata that have selected the *Exclude Literal* action and whose corresponding literals evaluate to **0**. By merely switching from *Exclude Literal* to *Include Literal* for a single one of these, our goal is achieved. That is, since we are dealing with conjunctive clauses, simply including a single literal that evaluates to **0** makes the whole conjunction also evaluate to **0**. In this manner, we guide the Tsetlin Automata towards eliminating False Positive Output.

	Truth Value of Target Clause C_j^i	**1**		**0**	
	Truth Value of Target Literal $x_k/\neg x_k$	**1**	**0**	**1**	**0**
Include Literal ($k \in I_j^i / k \in \bar{I}_j^i$)	P(Reward)	0	NA	0	0
	P(Inaction)	1.0	NA	1.0	1.0
	P(Penalty)	0	NA	0	0
Exclude Literal ($k \notin I_j^i / k \notin \bar{I}_j^i$)	P(Reward)	0	0	0	0
	P(Inaction)	1.0	0	1.0	1.0
	P(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 $x_k/\neg x_k$ in the clause C_j^i. Type II Feedback is triggered to decrease the number of clauses that incorrectly evaluates to **1** for a given input X.

Together, Type I Feedback and Type II Feedback interact to reduce the output error rate to a minimal level.

3.5.4 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, $(\hat{X}, \hat{y}^i) \in \mathcal{S}$, as input. Based on this, it produces a propositional formula in conjunctive normal form, $\Phi^i(X)$, for predicting the output, y^i.

We will now take a closer look at the algorithm, line-by-line.

Algorithm 1 The Tsetlin Machine

Input Training data $(\hat{X}, \hat{y}^i) \in \mathcal{S} \sim P(X, y^i)$, Number of clauses m, Output index i, Number of inputs o, Precision s, Threshold T

Output Completely trained conjunctive clauses $C_j^i \in \mathcal{C}^i$ for y^i

function TrainTsetlinMachine($\mathcal{S}, m, i, o, s, T$)
    $\mathcal{G}_1^i, \ldots, \mathcal{G}_m^i \leftarrow$ ProduceTsetlinMachine(m,o) ▷ Produce $2o$ TsetlinAutomata (TA) for each clause C_j^i, assigning TA_{2k-1}^{ij} to x_k and TA_{2k}^{ij} to $\neg x_k$. Both TA_{2k-1}^{ij} and TA_{2k}^{ij} belong to $\mathcal{G}_j^i$.
    $\mathcal{G}^i \leftarrow \{\mathcal{G}_1^i, \ldots, \mathcal{G}_m^i\}$ ▷ Collect each team $\mathcal{G}_j^i$ in $\mathcal{G}^i$
    repeat
        $\hat{X}, \hat{y} \leftarrow$ GetNextTrainingExample($\mathcal{S}$) ▷ Mini-batches, random selection, etc.
        $\mathcal{C}^i \leftarrow$ ObtainConjunctiveClauses($\mathcal{G}^i$) ▷ The Tsetlin Automata Teams $\mathcal{G}_j^i \in \mathcal{G}^i$ make their decisions, producing the conjunctive clauses.
        for $j \leftarrow 1, 3, \ldots, m-1$ **do** ▷ Provide feedback for clauses with positive polarity.
            if $\hat{y}^i = \mathbf{1}$ **then**
                if Random() $\leq \frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}^i(\hat{X}))))}{2T}$ **then**
                    TypeIFeedback($\mathcal{G}_j^i$) ▷ Output $\hat{y}^i = \mathbf{1}$ activates Type I Feedback for clauses with positive polarity.
                end if
            else if $\hat{y}^i = \mathbf{0}$ **then**
                if Random() $\leq \frac{T + \max(-T, \min(T, f_{\sum}(\mathcal{C}^i(\hat{X}))))}{2T}$ **then**
                    TypeIIFeedback($\mathcal{G}_j^i$) ▷ Output $\hat{y}^i = \mathbf{0}$ activates Type II Feedback for clauses with positive polarity.
                end if
            end if
        end for
        for $j \leftarrow 2, 4, \ldots, m$ **do** ▷ Provide feedback for clauses with negative polarity.
            if $\hat{y}^i = \mathbf{1}$ **then**
                if Random() $\leq \frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}^i(\hat{X}))))}{2T}$ **then**
                    TypeIIFeedback($\mathcal{G}_j^i$) ▷ Output $\hat{y}^i = \mathbf{1}$ activates Type II Feedback for clauses with negative polarity.
                end if
            else if $\hat{y}^i = \mathbf{0}$ **then**
                if Random() $\leq \frac{T + \max(-T, \min(T, f_{\sum}(\mathcal{C}^i(\hat{X}))))}{2T}$ **then**
                    TypeIFeedback($\mathcal{G}_j^i$) ▷ Output $\hat{y}^i = \mathbf{0}$ activates Type I Feedback for clauses with negative polarity.
                end if
            end if
        end for
    until StopCriteria($\mathcal{S}, \mathcal{C}^i$) $= \mathbf{1}$
    return PruneAllExcludeClauses($\mathcal{C}^i$) ▷ Return completely trained conjunctive clauses $C_j^i \in \mathcal{C}^i$ for y^i, after pruning clauses where all literals have been excluded.
end function

Algorithm 2 Type I Feedback - Combating False Negative Output

Input Input X, Clause C_j^i, Tsetlin Automata team $\mathcal{G}_j^i$, Number of inputs o

procedure GenerateTypeIFeedback($X, X_j^i, \mathcal{G}_j^i, o$)
  for $k \leftarrow 1, o$ **do** ▷ Reward/Penalize all Tsetlin Automata in $\mathcal{G}_j^i$.
    $x_k \leftarrow X[k]$
    if Random() $\leq$ TypeIFeedback(Reward, Action(TA_{2k-1}^{ij}), x_k, $C_j^i(X)$) **then**
      Reward(TA_{2k}^{ij}) ▷ Reward TA controlling x_k.
    else if Random() $\leq$ TypeIFeedback(Penalty, Action(TA_{2k-1}^{ij}), x_k, $C_j^i(X)$) **then**
      Penalize(TA_{2k}^{ij}) Reward(TA_{2k}^{ij}) ▷ Penalize TA controlling x_k.
    end if
    if Random() $\leq$ TypeIFeedback(Reward, Action(TA_{2k}^{ij}), $\neg x_k$, $C_j^i(X)$) **then**
      Reward(TA_{2k}^{ij}) ▷ Reward TA controlling $\neg x_k$.
    else if Random() $\leq$ TypeIFeedback(Penalty, Action(TA_{2k}^{ij}), $\neg x_k$, $C_j^i(X)$) **then**
      Penalize(TA_{2k}^{ij}) ▷ Penalize TA controlling $\neg x_k$.
    end if
  end for
end procedure

Algorithm 3 Type II Feedback - Combating False Positive Output

Input Input X, Clause C_j^i, Tsetlin Automata team $\mathcal{G}_j^i$, Number of inputs o

procedure GenerateTypeIIFeedback($X, X_j^i, \mathcal{G}_j^i, o$)
  for $k \leftarrow 1, o$ **do**
    $x_k \leftarrow X[k]$
    if Random() $\leq$ TypeIIFeedback(Penalty, Action(TA_{2k-1}^{ij}), x_k, $C_j^i(X)$) **then**
      Penalize(TA_{2k}^{ij}) ▷ Penalize TA controlling x_k.
    end if
    if Random() $\leq$ TypeIIFeedback(Penalty, Action(TA_{2k}^{ij}), $\neg x_k$, $C_j^i(X)$) **then**
      Penalize(TA_{2k}^{ij}) ▷ Penalize TA controlling $\neg x_k$.
    end if
  end for
end procedure

Lines 2-3. From the perspective of game theory, each Tsetlin Automaton, $\mathrm{TA}^{ij}_{2k}/\mathrm{TA}^{ij}_{2k-1}$, $j = 1, \ldots, m$, $k = 1, \ldots, o$, takes part in a large and complex game, consisting of multiple independent players. Every Tsetlin Automaton is assigned a user specified number of states, N, per action, for learning which action to perform. The start-up state is then randomly set to either N or $N + 1$. Each Tsetlin Automaton $\mathrm{TA}^{ij}_{2k}/\mathrm{TA}^{ij}_{2k-1}$ selects between two actions: Either to *include* or *exclude* a specific literal, $x_k/\neg x_k$, in a specific clause, C^i_j. These Tsetlin Automata are in turn organized into teams of $2o$ automata. Each team, $\mathcal{G}^i_j$, is responsible for a specific clause C^i_j, forming a sub-game.

Line 5. As seen in the algorithm, the learning process is driven by a set of training examples, $\mathcal{S}$, sampled from the joint input-output distribution $P(X, y^i)$, as described in Section 2. Each single training example $(\hat{X}, \hat{y}^i)$ is fed to the Tsetlin Machine, one at a time, facilitating online learning.

Line 6. In each iteration, the Tsetlin Automata decide whether to include or exclude each literal from each of the conjunctive clauses. The result is a new set of conjunctive clauses, $\mathcal{C}^i$, capable of predicting y^i.

Lines 7-17. The next step is to measure how well, $\mathcal{C}^i$, predicts the observed output $\hat{y}^i$ in order to provide feedback the Tsetlin Automata teams $\mathcal{G}^i_j$. As seen, feedback is generated directly based on the output of the summation function, $f_{\sum}(\mathcal{C}^i(X))$, from Eqn. 3.

In order to maximize pattern representation capacity, we use a threshold value T as target for the summation $f_{\sum}$. This mechanism is inspired by the finite-state automaton based resource allocation scheme for solving the knapsack problem in unknown and stochastic environments [7]. The purpose of the mechanism is to ensure that only a few of the available clauses are spent representing each specific sub-pattern. This is to effectively allocate sparse pattern representation resources among competing sub-patterns. To exemplify, assume that the correct output is $y^i = \mathbf{1}$ for an input X. If the votes accumulate to a total of T or more, neither rewards or penalties are provided to the involved Tsetlin Automata. This leaves the Tsetlin Automata unaffected.

Generating Type I Feedback. If the target output is $y^i = \mathbf{1}$, we randomly generate *Type I Feedback* for each clause $C^i_j \in \mathcal{C}^i$. The probability of generating Type I Feedback is:

$$\frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}^i)))}{2T}. \tag{5}$$

Generating Type II Feedback. If, on the other hand, the target output is $y^i = \mathbf{0}$, we randomly generate *Type II Feedback* for each clause $C^i_j \in \mathcal{C}^i$. The probability of generating Type II Feedback is:

$$\frac{T + \max(-T, \min(T, f_{\sum}(\mathcal{C}^i)))}{2T}. \tag{6}$$

Notice how the feedback vanishes as the number of triggering clauses correctly approaches $T/-T$. This is a crucial part of effective use of the available pattern representation capacity. Indeed, if the existing clauses already are able to capture the pattern $\hat{X}$ faced, there is no need to adjust any of the clauses.

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

Lines 18-28. After the clauses with positive polarity have received feedback. The next step is to invert the role of Type I and Type II Feedback, and feed the resulting feedback to the clauses with negative polarity.

Line 29. The above steps are iterated until a stopping criteria is fulfilled (for instance a certain number of iterations over the dataset), upon which the current clauses $\mathcal{C}^i$ are returned as the output of the algorithm.

As a final note, the resulting propositional formula, return by the algorithm, has been composed by the Tsetlin Automata with the goal of predicting the output y^i with optimal accuracy.

4 Theoretical Analysis

The reader may now have recognized that we have designed the Tsetlin Machine with mathematical analysis in mind, in order to facilitate a deeper understanding of our learning scheme. In this section, we argue that the Tsetlin Machine converges towards solving the problem of Pattern Learning with Propositional Logic from Definition 1 with probability arbitrary close to unity.

Assume that the propositional formula Φ^i for output y^i solves a given pattern recognition problem, per Definition 1. Let $\mathcal{X}^i = \{X|\Phi^i(X) = \mathbf{1}, X \in \mathcal{X}\}$ be the subset of input values X that makes Φ^i evaluate to $\mathbf{1}$. Furthermore, let $\overline{\mathcal{X}}^i = \{X|\Phi^i(X) = \mathbf{0}, X \in \mathcal{X}\}$ be the complement of $\mathcal{X}^i$. Let l_k be a literal referring to either x_k or $\neg x_k$. Now consider one of the conjunctive clauses, C^i_j, that takes part in Φ^i. Let $\mathcal{X}^i_j = \{X|C^i_j(X) = \mathbf{1}, X \in \mathcal{X}^i\}$ and let $\mathcal{X}^{i*}_j = \{X|C^i_j(X) = \mathbf{1} \wedge l_k = \mathbf{1}, X \in \mathcal{X}^i\}$.

Further, let the subgame involving the Tsetlin Automata that form the clause $C^i_j(X)$ be denoted by $\mathcal{G}^i_j = \{\mathrm{TA}^{ij}_k | 1 \leq k \leq 2o\}$. Consider one of the Tsetlin Automata $\mathrm{TA}^{ij}_k \in \mathcal{G}^i_j$ in the subgame $\mathcal{G}^i_j$. The payoffs it can receive are given in Table 2 and Table 3 for the whole range of subgame $\mathcal{G}^i_j$ outcomes, from the perspective of TA^{ij}_k. Finally, let (X, y^i) be an input-output pair sampled from $P(X, y^i)$.

Lemma 1. *The expected payoff of the action* Exclude Literal *within the subgame* $\mathcal{G}^i_j$ *is:*

$$\begin{aligned}P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{1}{s} + P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{1}{s} \quad - \\ P(X \in \mathcal{X}^{i*}_j | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{s-1}{s} - P(X \in \mathcal{X}^i_j \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{0})P(y^i = \mathbf{0}) \cdot 1.0 \end{aligned} \tag{7}$$

Proof. In brief, we receive an expected fractional *reward* $\frac{1}{s}$ every time y^i becomes $\mathbf{1}$, except when both l_k and $C^{i*}_j(X)$ in Φ^i evaluates to $\mathbf{1}$. In that case, we instead receive an expected fractional *penalty* of $\frac{s-1}{s}$ (see Table 2). Formally, we can reformulate this rewarding and penalizing as follows:

$$P(y^i = \mathbf{1} \wedge \neg(C^i_j(X) = \mathbf{1} \wedge l_k = \mathbf{1})) \cdot \frac{1}{s} - P(y^i = \mathbf{1} \wedge C^i_j(X) = \mathbf{1} \wedge l_k = \mathbf{1}) \cdot \frac{s-1}{s} = \tag{8}$$

$$P(y^i = \mathbf{1} \wedge \neg(X \in \mathcal{X}^{i*}_j)) \cdot \frac{1}{s} - P(y^i = \mathbf{1} \wedge X \in \mathcal{X}^{i*}_j) \cdot \frac{s-1}{s} = \tag{9}$$

$$P(y^i = \mathbf{1} \wedge X \in \overline{\mathcal{X}}^{i*}_j) \cdot \frac{1}{s} - P(y^i = \mathbf{1} \wedge X \in \mathcal{X}^{i*}_j) \cdot \frac{s-1}{s} = \tag{10}$$

$$\begin{aligned}P(y^i = \mathbf{1} \wedge X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j) \cdot \frac{1}{s} + P(y^i = \mathbf{1} \wedge X \in \overline{\mathcal{X}}^i) \cdot \frac{1}{s} \quad - \\ P(y^i = \mathbf{1} \wedge X \in \mathcal{X}^{i*}_j) \cdot \frac{s-1}{s} =\end{aligned} \tag{11}$$

$$\begin{aligned}P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{1}{s} + P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{1}{s} \quad - \\ P(X \in \mathcal{X}^{i*}_j | y^i = \mathbf{1})P(y^i = \mathbf{1}) \cdot \frac{s-1}{s}\end{aligned} \tag{12}$$

Furthermore, selecting the *Exclude Literal* when $l_k = \mathbf{0}$ and $y^i = \mathbf{0}$, while $C^i_j(X)$ evaluates to $\mathbf{1}$, provides a full *penalty* (see Table 3). As further seen in the table, false positive output never triggers penalties or rewards for the *Include Literal* action. All this is by design in order to suppress the output $\mathbf{1}$ from $C^i_j(X)$ in such cases. In other words, an even stronger Nash equilibrium exists when switching from *Include Literal* to *Exclude Literal* increases the probability of false positive output. This additional effect which enforces the Nash equilibrium can be formalized as follows:

$$P(y^i = \mathbf{0} \wedge l_k = \mathbf{0} \wedge C^i_j(X) = \mathbf{1}) \cdot 1.0 \quad = \tag{13}$$

$$P(X \in \mathcal{X}^i_j \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{0})P(y^i = \mathbf{0}) \cdot 1.0 \tag{14}$$

□

Lemma 2. *The expected payoff of the action* Include Literal *within the subgame* $\mathcal{G}^i_j$ *is:*

$$P(X \in \mathcal{X}^{i*}_j | y^i = \mathit{1}) P(y^i = \mathit{1}) \cdot \frac{s-1}{s} \quad -$$
$$P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathit{1}) P(y^i = \mathit{1}) \cdot \frac{1}{s} + P(X \in \overline{\mathcal{X}}^i | y^i = \mathit{1}) P(y^i = \mathit{1}) \cdot \frac{1}{s} \tag{15}$$

Proof. We now simply establish that the expected feedback of action *Include Literal* is symmetric to the expected feedback of action *Exclude Literal*, apart from not being affected by Type II Feedback.

$$P(y^i = \mathbf{1} \wedge C^i_j(X) = \mathbf{1}) \cdot \frac{s-1}{s} - P(y^i = \mathbf{1} \wedge \neg C^i_j(X) = \mathbf{1}) \cdot \frac{1}{s} \quad = \tag{16}$$
$$P(y^i = \mathbf{1} \wedge C^i_j(X) = \mathbf{1} \wedge l_k = \mathbf{1}) \cdot \frac{s-1}{s} - P(y^i = \mathbf{1} \wedge \neg(C^i_j(X) = \mathbf{1} \wedge l_k = \mathbf{1})) \cdot \frac{1}{s} \quad = \tag{17}$$
$$P(y^i = \mathbf{1} \wedge X \in \mathcal{X}^{i*}_j) \cdot \frac{s-1}{s} - P(y^i = \mathbf{1} \wedge X \in \overline{\mathcal{X}}^{i*}_j) \cdot \frac{1}{s} \tag{18}$$

As seen above, we verify symmetry by observing that the action *Include Literal* with $C^i_j(X) = \mathbf{1}$ enforces $l_k = \mathbf{1}$. This in turn allows us to replace $C^i_j(X) = \mathbf{1}$ with $X \in \mathcal{X}^i_j$. Together with the obvious symmetry of Table 3, this makes the expected payoff symmetric to that of *Exclude Literal* (apart from the Type II Feedback) .

In other words, for the same reasons that *Exclude Literal* has a negative expected payoff, *Include Literal* has a positive one, and vice versa! *This symmetry is by design to facilitate robust and fast learning in the game.* □

Theorem 1. *Let* Φ^i *be a solution to a pattern recognition problem per Definition 1. Then every clause* C^i_j *in* Φ^i *is a Nash equilibrium in the associated Tsetlin Machine subgame* $\mathcal{G}^i_j$*.*

Proof. We now have two scenarios. Either the literal l_k is part of the clause C^i_j in the solution Φ^i (the action selected by TA^{ij}_k is *Include Literal*). Otherwise, the literal is not part of the clause C^i_j in the solution Φ^i (the selected action is *Exclude Literal*). We will now consider each of these scenarios for Φ^i, and verify that both scenarios qualify as Nash equilibria.

Scenario 1: Literal included. A literal l_k is included in a clause C^i_j for two reasons:

1. To provide a certain granularity $\frac{1}{s}$ of C^i_j, i.e., to maintain $P(C^i_j(X)) = \frac{1}{s} + \epsilon$ for as small as possible $\epsilon > 0$.
2. To combat false positive output $C^i_j(X) = \mathbf{1}$ when $y = \mathbf{0}$, by including literals l_k that makes $C^i_j(X)$ evaluate to $\mathbf{0}$ whenever $y = \mathbf{0}$.

Consider the expected payoff of *Exclude Literal*:

$$P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{1}{s} + P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{1}{s} \quad -$$
$$P(X \in \mathcal{X}^{i*}_j | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{s-1}{s} - P(X \in \mathcal{X}^i_j \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{0}) P(y^i = \mathbf{0}) \cdot 1.0 \tag{19}$$

Let us first consider the situation where $P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1}) P(y^i = \mathbf{1})$ is zero. This means that we only obtain $y^i = \mathbf{1}$ when the input belongs to $\mathcal{X}^i$. Then we only need to consider:

$$P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{1}{s} \quad -$$
$$P(X \in \mathcal{X}^{i*}_j | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{s-1}{s} - P(X \in \mathcal{X}^i_j \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{0}) P(y^i = \mathbf{0}) \cdot 1.0 \tag{20}$$

Since $P(X \in \mathcal{X}^{i*}_j | y^i = \mathbf{1}) > \frac{1}{s}$ and $P(X \in \mathcal{X}^i \setminus \mathcal{X}^{i*}_j | y^i = \mathbf{1}) < \frac{s-1}{s}$ by definition, the expected payoff for *Exclude Literal* is negative for this reason alone. Another important task of $\mathrm{TA}^{ij}_k \in \mathcal{G}^i_j$

is to combat false positive output, which happens when y^i is $\mathbf{0}$ while $C_j^i(X)$ evaluates to $\mathbf{1}$. To achieve this, selecting the *Exclude Literal* when $l_k = \mathbf{0}$ and $y^i = \mathbf{0}$, while $C_j^{i*}(X)$ evaluates to $\mathbf{1}$, triggers a full *penalty* (see Table 3). In other words, the Nash equilibrium is further "strengthened" by potential false positive output.

Let us now return to the more general case of $\alpha = P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1})$ being greater than or equal to zero. Then we have $P(X \in \mathcal{X}_j^{i*} | y^i = \mathbf{1}) > (1-\alpha) \cdot \frac{1}{s}$ and $P(X \in \mathcal{X}^i \setminus \mathcal{X}_j^{i*} | y^i = \mathbf{1}) < (1-\alpha) \cdot \frac{s-1}{s}$ by definition. Additionally, we now alse get the expected *reward* $\alpha \cdot \frac{1}{s}$. The latter reward shifts the equilibrium towards *Exclude Literal*, which can be compensated for by artificially increasing s, to keep the expected payoff for *Exclude Literal* negative. *In other words, by manipulating s we can achieve the intended Nash equilibrium, even with extreme noise.*

Recall that the whole purpose of the above Nash equilibrium is to make sure that the patterns captured by the clause C_j^i is of an appropriate granularity, decided by s, finely balancing *Exclude Literal* actions against *Include Literal* actions. This is combined with the combating of false positive output through targeted selection of *Include Literal* actions.

To conclude, due to the established symmetry in payoff, switching from *Include Literal* to *Exclude Literal* leads to a net loss in expected payoff, providing a Nash equilibrium for the action *Include Literal.*

Scenario 2: Literal excluded. A literal l_k is excluded from a clause C_j^i for one reason: To maintain $P(C_j^i(X)) = \frac{1}{s} + \epsilon$ for as small as possible $\epsilon > 0$, i.e., to maintain a certain granularity for the patterns matched by C_j^i. Again, consider the expected payoff of *Exclude Literal* when $P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1})$ is zero:

$$\begin{aligned} & P(X \in \mathcal{X}^i \setminus \mathcal{X}_j^{i*} | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{1}{s} \quad - \\ P(X \in \mathcal{X}_j^{i*} | y^i = \mathbf{1}) P(y^i = \mathbf{1}) \cdot \frac{s-1}{s} & - P(X \in \mathcal{X}_j^i \setminus \mathcal{X}_j^{i*} | y^i = \mathbf{0}) P(y^i = \mathbf{0}) \cdot 1.0 \end{aligned} \tag{21}$$

We know that $P(X \in \mathcal{X}_j^i \setminus \mathcal{X}_j^{i*} | y^i = \mathbf{0}) P(y^i = \mathbf{0}) \cdot 1.0$ is minimal since Φ^i is a solution. For the same reason, we know that $P(X \in \mathcal{X}_j^{i*} | y^i = \mathbf{1}) \leq \frac{1}{s}$ if $P(X \in \mathcal{X}_j^i | y^i = \mathbf{1}) = \frac{1}{s} + \epsilon$. This again, implies that $P(X \in \mathcal{X}^i \setminus \mathcal{X}_j^{i*} | y^i = \mathbf{1}) > \frac{s-1}{s}$. In other words, the expected payoff of *Exclude Literal* is positive, which due to the symmetry again means that *Include Literal* has a negative expected payoff. For the case where $P(X \in \overline{\mathcal{X}}^i | y^i = \mathbf{1})$ is greater than zero, we can apply the same reasoning as in Scenario 1. Hence, the Nash equilibrium! $\square$

Theorem 2 (A conjunctive C_j^i that is not part of the solution Φ^i is not a Nash equilibrium)**.**

Proof. This theorem follows from the proof for Theorem 1. By invalidating any of the required conditions that made a clause C_j^i a Nash equilibrium, it can no longer be a Nash equilibrium. $\square$

Theorem 3. *The Tsetlin Machine converges to a solution Φ^i with probability arbirtrarily close to unity.*

Proof. Here follows a sketch for a proof. Any solution scheme that is capable of finding a single Nash equilibrium in a game will be able to solve each subgame $\mathcal{G}_j^i$. Each subgame $\mathcal{G}_j^i$ is played independently of the other subgames, apart from the indirect interaction through the summation function $f_{\sum}$. However, the feedback that connects the subgames only control how often each game is activated. Indeed, a subgame is activated with probability

$$\frac{T - \max(-T, \min(T, f_{\sum}(\mathcal{C}^i)))}{2T} \tag{22}$$

for Type I Feedback, and with probability:

$$\frac{T + \max(-T, \min(T, f_{\sum}(\mathcal{C}^i)))}{2T} \tag{23}$$

for Type II Feedback. These together merely control the mixing factor of the two different types of feedback, as well as the frequency with which the subgame is played. Type II Feedback is self-defeating, eliminating itself by nature to a minimum. As an equilibrium is found in each subgame, eventually, all subgames are stopped playing, i.e. $f_{\sum}(\mathcal{C}^i)$ always evaluates to either T or $-T$, and the Tsetlin Machine Game has been solved.

The Tsetlin Automaton is one particularly robust mechanism for solving games with common payoff, converging to a Nash equilibrium with probability arbitrarily close to unity. $\square$

5 Empirical Results

In this section we evaluate the Tsetlin Machine empirically using four diverse datasets[4]:

- **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 is designed to be a challenging machine learning benchmark, involving 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 so-called Curse of Dimensionality [26]. To examine robustness towards noise we have further randomly inverted 40% of the outputs.

For these datasets, we performed ensembles of 100 to 1000 independent replications with different random number streams. We did this to minimize the variance of the reported results, to avoid being deceived by random variation, and to provide the foundation for a statistical analysis of the merits of the different schemes evaluated.

Together with the Tsetlin Machine, we have also evaluated several classical machine learning techniques using the same random number streams. This includes Neural Networks, the Naive Bayes Classifier, Support Vector Machines, and Logistic Regression. Where appropriate, the different schemes were optimized by means of hyper-parameter grid searches. As an example, Figure 6 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. Note that we have not included 95% confidence intervals in the plot because our purpose was to find the s value that provided the best accuracy on *average*. We provide detailed results on random variation of the mean accuracy in each of the following experiments.

5.1 The Binary Iris Dataset

We first evaluated the Tsetlin Machine on the classical Iris dataset[5]. 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 increased the challenge by transforming the four input values into one consecutive sequence of 16 bits, four bits per float. It was thus necessary to also learn how to segment the

[4] The data sets can be found at `https://github.com/cair/TsetlinMachine`.

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

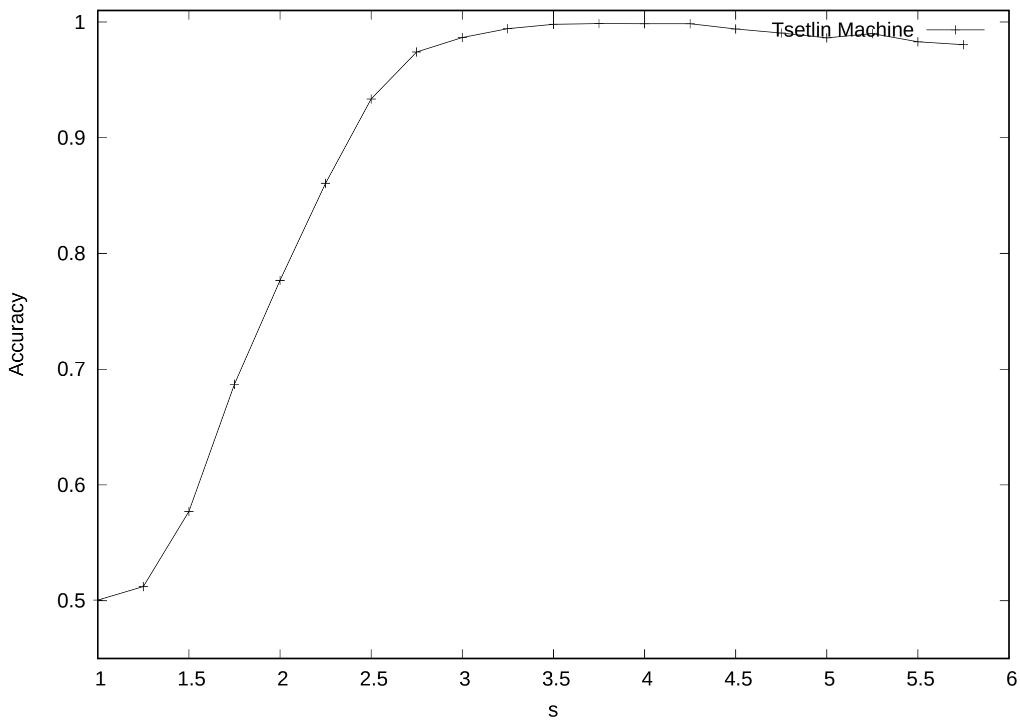

Figure 6: The mean accuracy of the Tsetlin Machine (y-axis) on the Noisy XOR Dataset for different values of the parameter s (x-axis).

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

We partitioned this dataset into a training set and a test set, with 80 percent of the data being used for training. We here randomly produced 1000 training and test data partitions. For each ensemble, we also randomly reinitialized the competing algorithms, to gain information on stability and robustness. The results are reported in Table 4.

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
Neural 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 4: The Binary Iris Dataset – accuracy on test data.

The Tsetlin Machine[6] used here employed 300 clauses, and used an s-value of 3.0 and a threshold T of 10. Furthermore, the individual Tsetlin Automata each had 100 states. This Tsetlin Machine was run for 500 epochs, and it is the accuracy after the final epoch that is reported. Even better configurations in terms of test accuracy were 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 used the last configuration produced, independently of whether the accuracy was good or not.

[6]In this experiment, we used a Multi-Class Tsetlin Machine, described in Section 6.1. We also applied Boosting of True Positive Feedback to Include Literal actions as described in Section 3.5.2.

In Table 4, 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 provided the highest mean accuracy. However, for the 95 %ile scores, most of the schemes obtained 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 high accuracy, when repeating the experiment 1000 times. Since the test set is merely a sample of the corresponding real-world problem, it is reasonable to assume that higher average accuracy translates to more robust performance overall.

The training set, on the other hand, was larger, and thus revealed more subtle differences between the schemes. The results obtained on the training set are shown in Table 5. As seen, the SVM here provided the better performance on average, while the Tsetlin Machine provided the second best accuracy. 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
Neural Networks	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 5: The Binary Iris Dataset – accuracy on training data.

5.2 The Binary Digits Dataset

We next evaluated the Tsetlin Machine on the classical Pen-Based Recognition of Handwritten Digits Dataset[7]. The original dataset consists of 250 handwritten digits from 44 different writers, for a total number of 10992 instances. We increased the challenge by removing the 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 The Binary Digits Dataset. Again we partitioned the dataset into training and test sets, with 80 percent of the data being used for training.

The Tsetlin Machine[8] used here contained 1000 clauses, used an s-value of 3.0, and had a threshold T of 10. Furthermore, the individual Tsetlin Automata each had 1000 states. The Tsetlin machine was run for 300 epochs, and it is the accuracy after the final epoch that is reported here.

Table 6 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 in the table, the Tsetlin Machine again clearly provided the highest accuracy on average, also when taking the 95% confidence intervals into account. Here the Tsetlin Machine were 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.

Performing poor on the test data and well on the training data indicates susceptibility to overfitting. Table 7 reveals that the other techniques, apart from the Naive Bayes Classifier, performed significantly better on the training data, unable to transfer this performance to the test data.

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

[8]In this experiment, we used a Multi-Class Tsetlin Machine, described in Section 6.1. We also applied Boosting of True Positive Feedback to Include Literal actions as described in Section 3.5.2.

Figure 7: The *Axis & Allies* mini game.

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*[9]. We designed this dataset to exhibit intricate pattern structures, involving optimal move prediction in a minimalistic, yet subtle, subgame of *Axis & Allies*. Indeed, superhuman performance for the *Axis & Allies* board game has not yet been attained. 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 7. Player 1 owns the "Caucasus" territory in the figure, while Player 2 owns "Ukraine" and "West Russia". At start-up, each player is randomly assigned 0-10 tanks and 0-20 infantry each. These units are their respective starting forces. For Player 2, they are randomly distributed among his two territories. The game consists of two rounds. First Player

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

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
Neural Networks	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 6: The Binary Digits Dataset – accuracy on test data.

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
Neural Networks	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 7: The Binary Digits Dataset – accuracy on training data.

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, merely 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 8: 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 8. 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 used 25% of the data for training, and 75% for testing, randomly producing 100 different partitions of the dataset. The Tsetlin Machine employed here contained 10 000 clauses, and used an s-value of 40.0 and a threshold T of 10. Furthermore, the individual Tsetlin Automata each had 1000 states. The Tsetlin machine was run for 200 epochs, and it is the accuracy after the final epoch that is reported.

Table 9 reports the results from predicting output bit 5 among the 20 output bits (we used 5 bits to represent each target value). 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 Machine and the neural network were capable of properly handling the complexity of the dataset, providing statistically similar performance. The Tsetlin Machine was quite stable performance-wise, while the neural network performance varied more.

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
Neural Networks	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 9: The *Axis & Allies* Dataset – accuracy on test data.

However, the number of clauses needed to achieve the above performance was 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 needed an s-value as large as 40, apparently, some of the pertinent sub-patterns must have been 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 10 reports performance on the training data. Random Forest distinguished itself by almost perfect predictions for the training data, thus clearly overfitting, but still performing well on the test set. The other techniques provided slightly improved performance on the training data, as expected.

5.4 The Noisy XOR Dataset with Non-informative Features

We now turn to the final dataset, which is an artifical one, constructed to uncover "blind zones" caused by XOR-like relations. Furthermore, the dataset contained a large number of random non-informative features to measure susceptibility towards the so-called Curse of Dimensionality [26]. To examine robustness towards noise, we have further randomly inverted 40% of the outputs.

The dataset consisted 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 were completely random. The two remaining inputs, however, were related to the output y through an XOR-relation, $y = XOR(x_{k_1}, x_{k_2})$. Finally, 40% of the outputs were inverted. Table 11 shows four examples from the dataset, demonstrating the high level of noise. We partitioned the dataset into training and test data, using 50% of the data for training.

The Tsetlin Machine[10] used here contained 20 clauses, and used an s-value of 3.9 and a threshold T of 15. Furthermore, the individual Tsetlin Automata each had 100 states. The Tsetlin Machine was run for 200 epochs, and it is the accuracy after the final epoch, that we report here.

Table 12 contains four of the clauses produced by the Tsetlin Machine. Notice how the noisy dataset from Table 11 has been turned into informative propositional formulas that capture the structure of the dataset.

The empirical results are found in Table 13. 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, made 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, saw through

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

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
Neural Networks	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 10: The *Axis & Allies* Dataset – accuracy on training data.

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 11: The Noisy XOR Dataset with Non-informative Features.

No.	**Sign**	**Clause Learned**
1	$+$	$\neg x_1 \land x_2$
2	$-$	$\neg x_1 \land \neg x_2$
3	$+$	$x_1 \land \neg x_2$
4	$-$	$x_1 \land x_2$

Table 12: Example of four clauses composed by the Tsetlin Machine for the XOR Dataset with Non-informative Features.

both the XOR-relationship and the noise, capturing the underlying pattern. SVM performed slightly better than the Naive Bayes Classifier and Logistic Regression, however, was distracted by the random 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
Neural Networks	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 13: The Noisy XOR Dataset with Non-informative Features – accuracy on test data.

Figure 8 shows how accuracy degraded with less data, when we varied the dataset size from 1000 examples to 20 000 examples. As expected, Naive Bayes and Logistic Regression guessed blindly for all the different data sizes. The main observation, however, is that the accuracy advantage the Tsetlin Machine had over neural networks increased with less training data. Indeed, it turned out that the Tsetlin Machine performed robustly with small training data sets in all of our experiments.

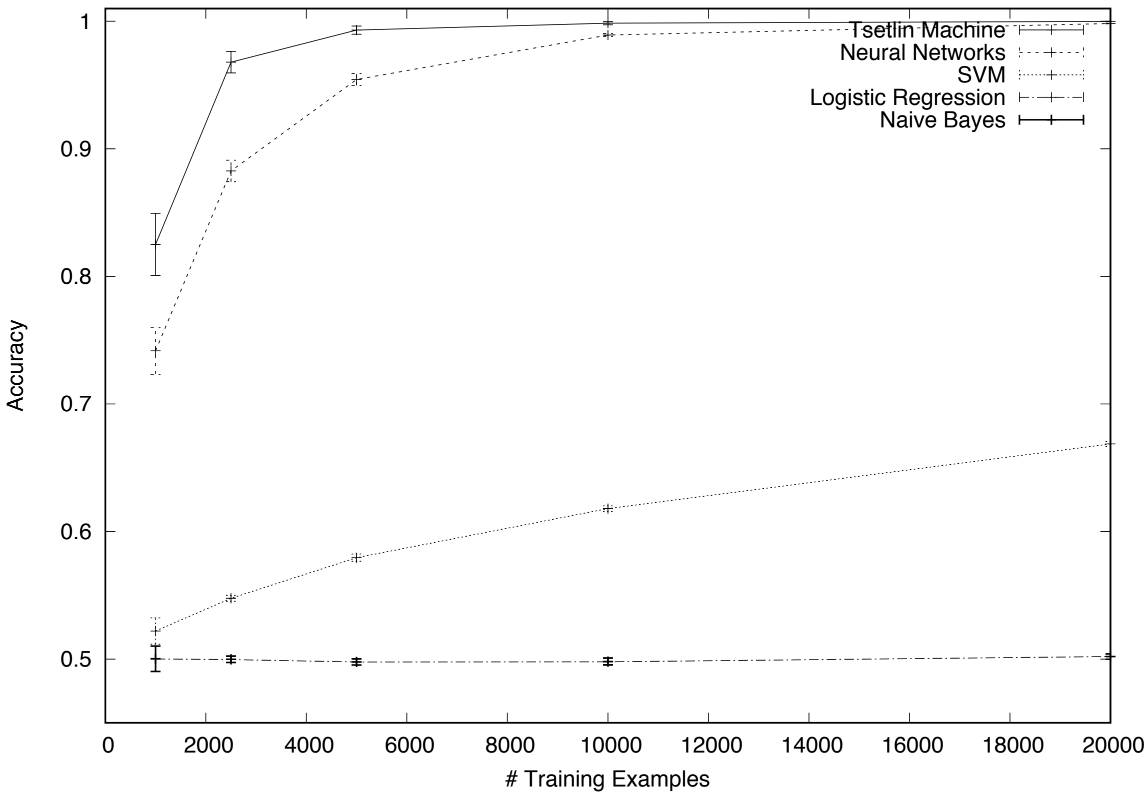

Figure 8: Accuracy (y-axis) for the Noisy XOR Dataset for different training dataset sizes (x-axis).

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 present the methodology for connecting multiple Tsetlin Machines in more advanced architectures. For demonstration purposes, we will use four different example 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, $y \in \{1, \ldots, n\}$. Such a multi-class pattern recognition problem can be handled by the Tsetlin Machine by representing y as bits, using multiple outputs, y_i. In this section, however, we present an alternative architecture that addresses the multi-class pattern recognition problem more directly.

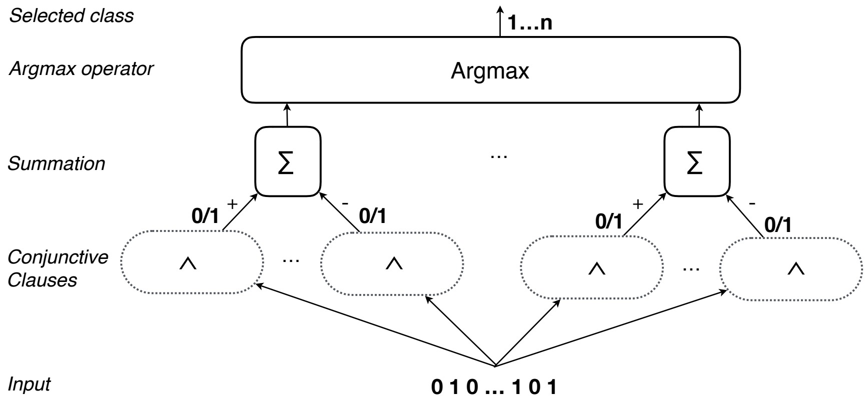

Figure 9: The Multi-Class Tsetlin Machine.

Figure 9 depicts the Multi-Class Tsetlin Machine[11] which replaces the threshold function of each output $y^i, i = 1, \ldots, n$ with a single **argmax** operator. With the **argmax** operator, the index i of the largest sum $f_{\sum}(\mathcal{C}^i(X))$ is outputted as the final output of the Tsetlin Machine:

$$y = \text{argmax}_{i=1,\ldots,n}(f_{\sum}(\mathcal{C}^i(X))). \tag{24}$$

In this manner, each propositional formula Φ^i, consisting of clauses $\mathcal{C}^i$, captures the pertinent aspects of the respective class i that it models.

Training is done as described in Section 3.5, apart from one critical modification. Assume we have $\hat{y} = i$ for the current observation, $(\hat{X}, \hat{y})$. Then the Tsetlin Automata team behind $\mathcal{C}^i$ is trained as per $\hat{y}^i = 1$ in the original Algorithm 1. Additionally, a random class $q \neq i$ is selected. The Tsetlin Automata team behind $\mathcal{C}^q$ is then trained in accordance with $\hat{y}^i = 0$ in the original algorithm (trained with opposite feedback, i.e., Type I Feedback becomes Type II Feedback, and vice versa).

6.2 The Fully Connected Deep Tsetlin Machine

Another architectural family is the Fully Connected Deep Tsetlin Machine [27], illustrated in Figure 10. 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^i, exactly as described in Section 3.5. The training procedure is thus similar to the strategy Hinton et al. used to train their pioneering Deep Belief Networks, layer-by-layer, in 2006 [28]. 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.

[11] An implementation of the Multi-Class Tsetlin Machine can be found at `https://github.com/cair/TsetlinMachine`.

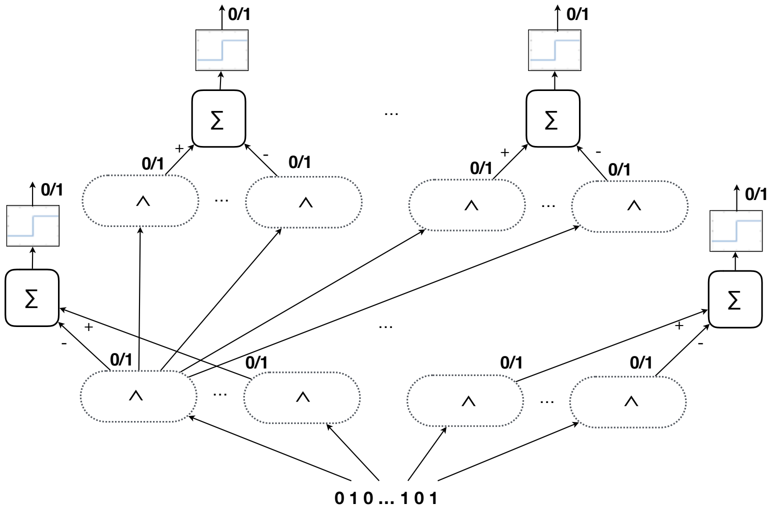

Figure 10: The fully connected Deep Tsetlin Machine.

6.3 The Convolutional Tsetlin Machine

We next demonstrate how self-contained and independent Tsetlin Machines can interact to build a Convolutional Tsetlin Machine [29], illustrated in Figure 11. The Convolutional Tsetlin

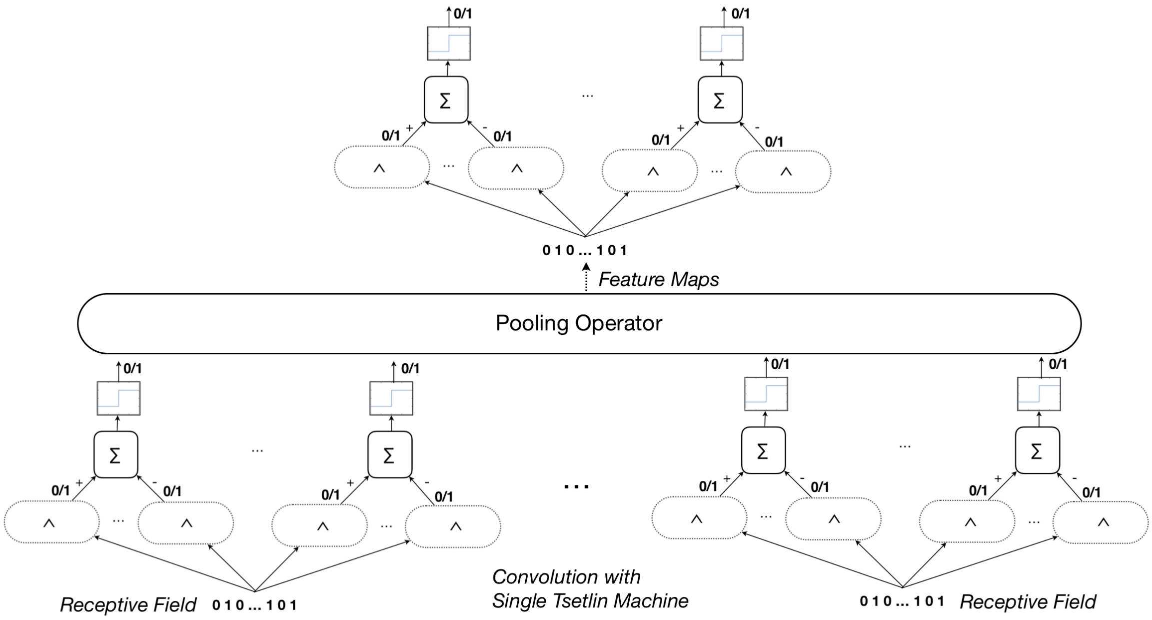

Figure 11: The Convolutional Tsetlin Machine.

Machine is a deep architecture based on mathematical convolution, akin to Convolutional Neural Networks [30, 31]. 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^i. 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^i, exactly as described in Section 3.5. 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 [32] (Figure 12). In all brevity, the same Tsetlin Machine is here reused from time step to time step. By taking the output from the clauses 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 [33]. Again, the architecture can be trained layer by

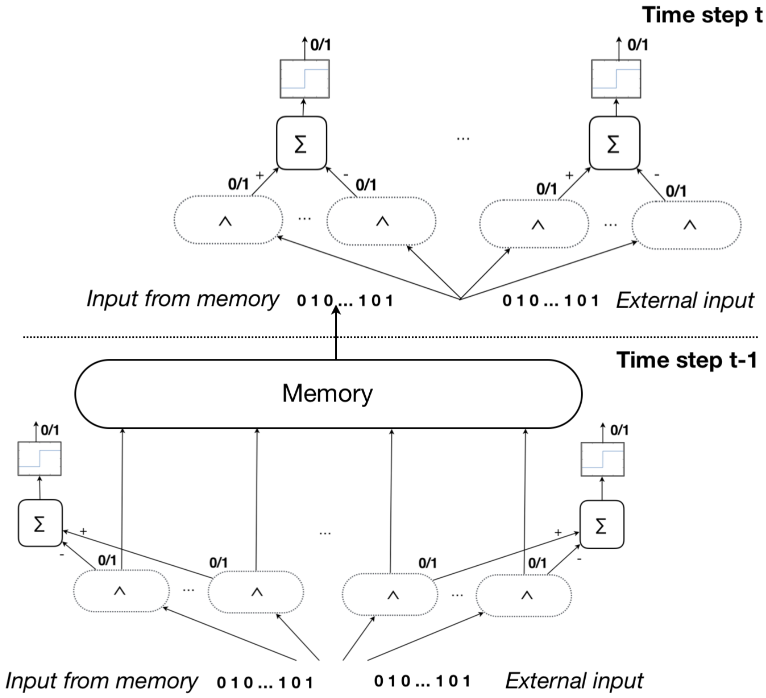

Figure 12: The Recurrent Tsetlin Machine.

layer, directly from the target output $y^i(t)$ 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 thousands 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 forms a set of sub-games where the Nash equilibria 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 four diverse pattern recognition problems. Overall, the Tsetlin Machine is particularly suited for digital computers, being merely based on simple bit manipulation with AND-, OR-, and NOT gates. Both input, hidden patterns, and output are expressed with easy-to-interpret bit patterns. In thorough empirical evaluations on four distinct benchmarks, the Tsetlin Machine outperformed both single layer neural networks, Support Vector Machines, Random Forests, the Naive Bayes Classifier and Logistic Regression. It further turns out that the Tsetlin Machine requires much less data than neural networks, even outperforming the Naive Bayes Classifier in data sparse environments.

The Tsetlin Machine is a completely new tool for machine learning. Based on its solid theoretical foundation in automata- and game theory, promising empirical results, and its ability to act as a building block in more advanced systems, we believe the Tsetlin Machine has the potential to impact the AI field as a whole, opening a wide range of research paths ahead. For instance, by demonstrating that the longstanding problem of vanishing signal-to-noise ratio can be solved, the Tsetlin Machine provides a new game theoretic framework for recasting the problem of pattern recognition, which opens opportunities for further advancement. Apart from the basic Tsetlin Machine, the Multi-Class Tsetlin Machine, the Fully Connected Deep Tsetlin Machine, the Convolution Tsetlin Machine, and the Recurrent Tsetlin Machine architectures can be an attractive starting point for further exploration. Lastly, the high accuracy and robustness 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.

Code Availability

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

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. A. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, p. 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] J. Carroll, *Theory of Finite Automata With an Introduction to Formal Languages.* Prentice Hall, 1989.

[5] O.-C. Granmo and B. Oommen, "Solving stochastic nonlinear resource allocation problems using a hierarchy of twofold resource allocation automata," *IEEE Transactions on Computers*, vol. 59, no. 4, 2010.

[6] 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.

[7] 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.

[8] 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.

[9] A. Yazidi, O.-C. Granmo, B. John Oommen, and M. Goodwin, "A novel strategy for solving the stochastic point location problem using a hierarchical searching scheme," *IEEE Transactions on Cybernetics*, vol. 44, no. 11, pp. 2202 – 2220, 2014.

[10] B. J. Oommen, S.-W. Kim, M. T. Samuel, and O.-C. Granmo, "A solution to the stochastic point location problem in metalevel nonstationary environments," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 38, no. 2, pp. 466–476, 2008.

[11] O.-C. Granmo and N. Bouhmala, "Solving the Satisfiability Problem Using Finite Learning Automata," *International Journal of Computer Science and Applications*, vol. 4, no. 3, pp. 15–29, 2007.

[12] N. Bouhmala and O.-C. Granmo, "Combining Finite Learning Automata with GSAT for the Satisfiability Problem," *Engineering Applications of Artificial Intelligence (EAAI)*, vol. 23, no. 5, pp. 715–726, 2010.

[13] ——, "Stochastic Learning for SAT-Encoded Graph Coloring Problems," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 3, pp. 1–19, 2010.

[14] A. Yazidi, O.-C. Granmo, and B. Oommen, "Service selection in stochastic environments: A learning-automaton based solution," *Applied Intelligence*, vol. 36, no. 3, pp. 617–637, 2012.

[15] 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.

[16] O.-C. Granmo and B. Oommen, "Optimal sampling for estimation with constrained resources using a learning automaton-based solution for the nonlinear fractional knapsack problem," *Applied Intelligence*, vol. 33, no. 1, pp. 3–20, 2010.

[17] A. Yazidi, O.-C. Granmo, and B. Oommen, "Learning-Automaton-Based Online Discovery and Tracking of Spatiotemporal Event Patterns," *IEEE Transactions on Cybernetics*, vol. 43, no. 3, pp. 1118 – 1130, 2013.

[18] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equipartitioning problem," *Computers, IEEE Transactions on*, vol. 37, no. 1, pp. 2–13, 1988.

[19] 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, 7 2016.

[20] B. J. Oommen, S. Misra, and O.-C. Granmo, "Routing Bandwidth Guaranteed Paths in MPLS Traffic Engineering: A Multiple Race Track Learning Approach," *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 959–976, 2007.

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

[22] 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.

[23] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," 2001.

[24] T. M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[25] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, 1971.

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

[27] O.-C. Granmo, "The Recurrent Tsetlin Machine," *In Preparation*, 2018.

[28] 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.

[29] O.-C. Granmo, "The Convolutional Tsetlin Machine," *In Preparation*, 2018.

[30] 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.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 1, 2012, pp. 1097–1105.

[32] O.-C. Granmo, "The Fully Connected Deep Tsetlin Machine," *In Preparation*, 2018.

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