NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

THESIS

Distributed Online Learning of Probabilistic Logical Theories for Complex Event Recognition

Evangelos M. Neamonitis

Supervisors: **Panagiotis Stamatopoulos** Assistant Professor(N.K.U.A.),
Nikolaos Katzouris Associate Researcher(N.C.S.R. Demokritos)

ATHENS

SEPTEMBER 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατανεμημένη Online Μηχανική Μάθηση Πιθανοτικών Λογικών Θεωριών για Αναγνώριση Σύνθετων Γεγονότων

Ευάγγελος Μ. Νεαμονίτης

Επιβλέποντες: **Παναγιώτης Σταματόπουλος** Επικ. Καθηγητής(Ε.Κ.Π.Α.),
Νικόλαος Κατζούρης Συνεργαζόμενος Ερευνητής(Ε.Κ.Ε.Φ.Ε. Δημόκριτος)

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2020

BSc sTHESIS

Distributed Online Learning of Probabilistic Logical Theories for Complex Event Recognition

Evangelos M. Neamonitis
S.N.: 1115201400123

Supervisors: **Panagiotis Stamatopoulos** Assistant Professor(N.K.U.A.),
Nikolaos Katzouris Associate Researcher(N.C.S.R. Demokritos)

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατανεμημένη Online Μηχανική Μάθηση Πιθανοτικών Λογικών Θεωριών για Αναγνώριση Σύνθετων Γεγονότων

Ευάγγελος Μ. Νεαμονίτης
Α.Μ.: 1115201400123

Επιβλέποντες: Παναγιώτης Σταματόπουλος Επικ. Καθηγητής(Ε.Κ.Π.Α.),
Νικόλαος Κατζούρης Συνεργαζόμενος Ερευνητής(Ε.Κ.Ε.Φ.Ε. Δημόκριτος)

ABSTRACT

Complex Event Recognition (CER) systems detect occurrences of complex events (e.g meeting, moving, dangerous driving) in a streaming time-stamped input of simple events using known event patterns. Logic-based approaches that are able to learn Event Calculus theories are of particular interest in CER, as they can effectively deal with uncertainty and noise in data streams, thus being a robust alternative to the costly process of manually crafting event pattern theories. In this context WOLED has been presented [19]. WOLED is a system that is based on Answer Set Programming (ASP). In recent years, the amount of data produced has seen an unprecedented increase. CER systems, ought to be able to cope with this and scale to the need of a given application. We focus on ways to tackle this rising problem by attempting to perceive WOLED in a distributed learning scenario with multiple learners. In this study, we compare and evaluate different possible communication protocols for sharing learned complex event patterns between learners.

SUBJECT AREA: Complex Event Recognition

KEYWORDS: Inductive Logic Programming ,Event Calculus, Distributed Systems, Complex Event Recognition, Geometric Monitoring

ΠΕΡΙΛΗΨΗ

Τα Συστήματα Ανανώρισης Σύνθετων Γεγονότων εντοπίζουν συμβάντα σύνθετων γεγονότων (π.χ: συνάντηση, ομαδική κίνηση, επικίνδυνη οδήγηση κλπ) σε συγκεκριμένες χρονικές στιγμές σε ροή δεδομένων εισόδου, χρησιμοποιώντας γνωστά απλά γεγονότα. Συστήματα Αναγνώρισης σύνθετων γεγονότων βασιμένα στη λογική πρώτης τάξης που μαθαίνουν θεωρίες Λογισμού Γεγονότων (Event Calculus) παρουσιάζουν ιδιαίτερο ενδιαφέρον στον τομέα Αναγνώρισης Σύνθετων Γεγονότων, καθώς μπορούν να ανταπεξέλθουν αποτελεσματικά σε ροές δεδομένων με θόρυβο και αβεβαιότητα. Έτσι τα συστήματα αυτά είναι μια ισχυρή εναλλακτική στην κοστοβόρα διαδικασία της χειροκίνητης παραγωγής θεωριών από μοτίβα γεγονότων. Σε αυτό το πλαίσιο ο αλγόριθμος WOLED παρουσιάστηκε στο [19]. Ο WOLED είναι βασισμένος σε Answer Set Programming (ASP). Τα τελευταία χρόνια, έχει παρατηρηθεί μια πρωτοφανής αύξηση στην ποσότητα των παραγώμενων δεδομένων. Τα Συστήματα Αναγνώρισης Σύνθετων Γεγονότων θα πρέπει να ανταπεξέρχονται στην αύξηση αυτή και να προσαρμόζονται στις απαιτήσεις ζητούμενων εφαρμογών. Στην εργασία αυτή εστιάζουμε σε τρόπους προσαρμογής του WOLED σε ένα σενάριο κατανεμημένης μηχανικής μάθησης με πολλαπλούς κόμβους μάθησης (learners). Συγκρίνουμε και αξιολογούμε διαφορετικά προτόκολα επικοινωνίας για τη διαμοίραση μεταξύ των κόμβων των παραγόμενων μοτίβων σύνθετων γεγονότων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγνώριση Σύνθετων Γεγονότων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Επαγωγικός Λογικός Προγραμματισμός, Λογισμός Γεγονότων Κατανεμημένα, Συστήματα, Αναγνώριση Σύνθετων Γεγονότων, Γεωμετρική Επίβλεψη

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Complex event recognition (CER) systems [4] detect occurrences of specific complex events of interest when given as an input sequences of simple events written in the form of first-order logic rules. For example a CER system can recognize when two people are meeting by observing simpler data such as each person's orientation and the distance between them. Machine learning algorithms with the ability to either construct or update such patterns, are of great importance as their manual development is proven to be a difficult and time consuming task. Such algorithms, are most beneficial when working in an online learning fashion, using a set of Complex Event patterns for inference and labeled streaming data for updating the aforementioned Complex event set.

In many cases, events are represented by first-order rules that follow specific patterns, defined by a knowledge base. A number of Logic-based CER systems have been proposed in recent years([2], [3], [21]), that deal effectively with noise and uncertainty in data streams. [19] presented WOLED (Online Learning of Weighted Event Definitions), an algorithm based on answer set programming (ASP), that learns CE patterns in the form of weighted rules in the Event Calculus. WOLED was proven to be particularly efficient over other similar implementations. An obvious next step, is to adapt this system to real world application circumstances.

It is evident, that in recent years, with the massive quantity of data produced every moment,it is getting harder and harder for machine learning application executed in a single compute node to cope with real world circumstances, albeit their capabilities. As a result, the use of distributed learning systems are already a commonplace solution for big data applications. Distributed learning systems, can either split the data or the model to multiple compute nodes and thus significantly reducing the time needed even for computation-heavy learning processes.

The main purpose of this work, is to expand and utilize WOLED's effectiveness in a distributed learning scenario in which the dataset is split among multiple learners. Each learner, by processing input data, produces new Complex Event patterns, updates the weights of the already existing Complex Event pattern set and shares its updates with the rest of the learners. We try and evaluate different communication protocols for gathering all pattern sets to a communication hub so as to merge them into a global one that will be sent back to each learner in order to resume its learning process.

2. BACKGROUND AND RELATED WORK

2.1 Complex Event Recognition

2.1.1 First-order Logic

We assume a first-order language where atoms, literals (possibly negated atoms), rules and logic programs are defined as in [12] and not denotes negation as failure. Rules, atoms, literals and programs are ground if they contain no variables. Rules are denoted by $\alpha \leftarrow \delta_1, \ldots, \delta_n$, where α is an atom and $\delta_1, \ldots, \delta_n$ a conjunction of literals. An interpretation I is a set of true ground atoms. I satisfies a ground literal a (resp. not a) iff $a \in I$ (resp. $a \notin I$) and it satisfies a ground rule iff it satisfies the head, or does not satisfy the body. I is a minimal (Herbrand) model of a logic program Π iff it satisfies every ground rule in Π and none of its strict subsets has this property. I is an answer set of Π iff it is a minimal model of the program that results from the ground instances of Π, after removing all rules with a negated literal not satisfied by I, and all negative literals from the remaining rules.

2.1.2 The Event Calculus for CER

The Event Calculus [23] is a logic programming formalism for representing events and their effects. As mentioned in [27] : "The Event Calculus is a logical mechanism that infers what's true when given what happens when and what actions do. The what happens when part is a narrative of events, and the what actions do part describes the effects of actions."

. Its underlying ontology is made up of three main parts:

1. **Time points:** time is measured in time points, integers that denote a specific moment.
2. **Fluents:** also known as variables. They are properties that have a certain value in a given time point
3. **Events:** occurrences in time that may affect fluents and alter their value

Its axioms incorporate the commonsense law of inertia, according to which fluents persist over time, unless they are affected by an event. Its basic predicates and axioms are presented in Table 2.1(a), (b). Axiom (1) states that a fluent F holds at time T if it has been initiated at the previous time point, while Axiom (2) states that F continues to hold unless it is terminated. Definitions of initiatedAt/2 and terminatedAt/2 predicates are provided in a application-specific manner.

Using the Event Calculus in a CER context allows to reason with CEs that have duration in time and are subject to commonsense phenomena, via associating CEs to fluents. In this case, a set of CE patterns is a set of conditions that initiate/terminate a target CE, i.e., a set of initiatedAt/2 and terminatedAt/2 rules.

Table 2.1: (a), (b) The basic predicates and the EC axioms. (c) Example CAVIAR data. For example, at time point 1 person with *id₁* is *walking*, her (X, Y) coordinates are $(201, 454)$ and her direction is $270°$. The query atoms for time point 1 ask whether persons id_1 and id_2 are moving together at the next time point. (d) An example of two domain-specific axioms in the EC. E.g. the first rule dictates that *moving together* between two persons X and Y is initiated at time T if both X and Y are walking at time T, their euclidean distance is less than 25 pixel positions and their difference in direction is less than $45°$. The second rule dictates that *moving together* between X and Y is terminated at time T if one of them is standing still at time T and their euclidean distance at T is greater that 30.

(a)

Predicate	**Meaning**
happensAt(E, T)	Event E occurs at time T.
initiatedAt(F, T)	At time T, a period of time for which fluent F holds is initiated.
terminatedAt(F, T)	At time T, a period of time for which fluent F holds is terminated.
holdsAt(F, T)	Fluent F holds at time T.

(b)

The axioms of the Event Calculus

holdsAt$(F, T + 1) \leftarrow$ initiatedAt(F, T) $\quad (1)$

holdsAt$(F, T + 1) \leftarrow$ holdsAt(F, T), not terminatedAt(F, T) $\quad (2)$

(c)

Observations I_1 at time 1:

happensAt(*walk*(*id₁*), *1*)
happensAt(*walk*(*id₂*), *1*)
coords(*id₁*, *201*, *454*, *1*)
coords(*id₂*, *230*, *440*, *1*)
direction(*id₁*, *270*, *1*)
direction(*id₂*, *270*, *1*)

Target CE instances at time 1:

holdsAt$(move(id_1, id_2), 2)$
holdsAt$(move(id_2, id_1), 2)$

(d)

Weighted CE patterns:

1.234 initiatedAt$(move(X, Y), T) \leftarrow$
happensAt(*walk*(*X*), *T*),
happensAt(*walk*(*Y*), *T*),
close(*X*, *Y*, *25*, *T*),
orientation(*X*, *Y*, *45*, *T*)

0.923 terminatedAt$(move(X, Y), T) \leftarrow$
happensAt(*inactive*(*X*), *T*),
not *close*(*X*, *Y*, *30*, *T*)

2.1.3 The CAVIAR dataset

Figure 2.1: Frame from a video in the Caviar dataset. Yellow boxes refer to activities of a single person such as walking, active etc, whereas green boxes refer to activities that include multiple people such as meeting, moving together etc. People in the image that are not included in a box are not moving

The task of activity recognition, as defined in the CAVIAR project[1] is to detect occurances of CEs as combinations of simple events and additional domain knowledge, such as a person's position and direction. The CAVIAR dataset consists of videos taken in a public space, where actors perform given activities. These videos have been manually annotated by the CAVIAR team to provide the ground truth for two types of activity. The first type, corresponding to simple events, consists of knowledge about a person's activities at a certain video frame/time point (e.g. *walking, standing still* and so on). The second type, corresponding to CEs/fluents, consists of activities that involve more than one person, for instance two people *moving together, meeting each other* and so on.

Table 2.1(c) presents an example of CAVIAR data, consisting of *observations* for a particular time point, in the form of an interpretation I_1. A stream of interpretations is matched against a set of CE patterns (initiation/termination rules – see Table 2.1(d)), to infer the truth values of CE instances in time, using the Event Calculus axioms as a reasoning engine. We henceforth call the atoms corresponding to CE instances whose truth values are

[1] `http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/`

to be inferred/predicted, *target CE instances*. Table 2(c) presents the target CE instances corresponding to the observations in I_1. Note that at time t the corresponding target CE instances refer to $t+1$, in accordance to the Event Calculus axioms, which infer the truth value of a CE instance at a time point, base on what happens at the previous time point.

2.2 The structure & learning process of WOLED

2.2.1 Overview of the Learning process

The main purpose of WOLED is to learn the structure and weights of initiatedAt/2 and terminatedAt/2 CE patterns, while using their current version at each point in time to perform CER in the streaming input. The learning process that is concisely described by Algorithm 1 consists of the following steps:

1. At each time t the learner maintains a theory H_t (weighted CE pattern set, as in 2.1(c)). For each pattern in H_t the learner also stores predictive statistics namely: TPs (True Positives), TNs (True Negatives), FPs (False Positives) and FNs (False Negatives). The learner also has access to some static background knowledge B (e.g. the axioms of the Event Calculus – 2.1(a)) and receives an interpretation I_t (as in 2.1(b)), which is composed by a mini-batch of observations.
2. The learner performs inference on the interpretation I_t which results in a number of inferred holdsAt/2 instances of the target complex event predicate. This step is analogous to the prediction step of a traditional Machine Learning algorithm.
3. For all the predicates of the inferred state, the true value is revealed and the learner's mistakes are identified.
4. Mistakes in the inferred state are used by the learner to update the weights and statistics of the already existing CE patterns in H_t
5. The learner attempts to produce new CE patterns that combined with the updated weights of H_t give a new theory H_{t+1}

2.2.2 Weighted complex event patterns

In WOLED, the CE patterns included in a logic program Π are associated with real-valued weights, defining a probability distribution over answer sets of Π. An answer set of a program with weighted rules may satisfy subsets of these rules, and these rules' weights determine the answer set's probability. Based on this observation, [24] propose to assign probabilities to answer sets of a program Π with weighted rules as follows:

- For each interpretation I, first find the maximal subset R_I of the weighted rules in Π that are satisfied by I.

- Then, assign to I a weight $W_{\Pi}(I)$ proportional to the sum of weights of the rules in R_I, if I is an answer set of R_I, else assign zero weight.
- Finally, define a probability distribution over answer sets of Π by normalizing these weights.

Formally, let w_r be the weight of rule r and ans(Π) the set of all interpretations I which are answer sets of R_I and which, moreover, satisfy all hard-constrained rules in Π (rules without weights). Then

$$W_{\Pi}(I) = \begin{cases} \exp\left(\sum_{r \in R_I} w_r\right) & \text{if } I \in \text{ans}(\Pi) \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$P_{\Pi}(I) = \frac{W_{\Pi}(I)}{\sum_{J \in \text{ans}(\Pi)} W_{\Pi}(J)} \tag{2.2}$$

2.2.3 WOLED's learning process using ASP tools

In this section we describe WOLED's learning process steps in more detail using ASP tools.

2.2.3.1 Getting the inferred state

When an interpretation I_t ,formed by a mini-batch of observations , arrives, WOLED generates the inferred state (predicts detections of the given complex event) by using MAP (Maximum A Posteriori) probabilistic inference. This corresponds to finding a most probable answer set $\mathcal{A}$ of $\Pi = B \cup H_t \cup I_t$. It is evident from equations 2.1, 2.2 that

$$\mathcal{A} = \arg\max_{I \in \text{ans}(\Pi)} P_{\Pi}(I) = \arg\max_{I \in \text{ans}(\Pi)} W_{\Pi}(I) = \arg\max_{I \in \text{ans}(\Pi)} \sum_{r \in R_I} w_r \tag{2.3}$$

which means that a most probable answer set in one that maximizes the weights of satisfied rules.As this is a MaxSat problem it is assigned to an answer set solver. The inferred state is the number of target events in the optimal, answer set.

2.2.3.2 Weight learning

For weight learning, after getting the inferred state from the interpretation I_t, the available true state labels are revealed and the weights of the CE patterns in H_t are updated by comparing their true groundings in the inferred state and the revealed true state. A true

grounding of an initiatedAt/2 or terminatedAt/2 CE pattern r_i is a grounding of r_i such that holdsAt$(\alpha, t+1)$ is true for initiatedAt/s2 patterns and false for terminatedAtpatterns.

Using the AdaGrad algorithm [8], an alternate version of the Gradient Descent that dynamically adapts the promotion/demotion of learned patterns by considering each pattern's past performance (TPs, TNs, FPs, FNs). The weights of CE patterns, that contributed to getting correct predictions are increased while the weights of CE patterns giving erroneous predictions are decreased. This algorithm, updates a weight vector which in our case is a vector containing the weights of the CE patterns as in Table 2.1(d).

2.2.3.3 Learning New CE patterns

In order to expand the current theory H_t, WOLED tries to generate new initiatedAt./2 patterns from FN predictions and terminatedAt/2 patterns from FP predictions respectively. Generating new patterns, can prevent similar mistakes in future interpretations. To illustrate the possible need for new complex event patterns,a FN means that at time t a complex event instance was predicted as false where in reality it was labeled as true. This could have been prevented if there existed an initiatedAt/2 pattern for the target complex event at a time prior to t. Generating patterns from all mistakes in the inferred state, would result in very large theories at some point. Most of the rules added would not contribute to increase WOLED's predictive performance. In this context, WOLED follows this strategy for generating new CE patterns:

1. A set of bottom clauses (rules) [5] is generated using the constants of the mistakes in the inferred state to create new ground initiatedAt/2 and terminatedAt/2 atoms, which are placed in the head of a set of initially empty-bodied rules.
2. The bodies of these rules are then populated with literals, grounded with constants that appear in the head and are true in the current data interpretation I_t.
3. Constants in the bottom rules are replaced by variables and the bottom rule set is "compressed" to a bottom theory $H_\perp$, which consists of unique, w.r.t. θ-subsumption, variabilized bottom rules. The new CE patterns are chosen among those that θ-subsume $H_\perp$.
4. The generalization technique of [26], [20]), which allows to search into the space of theories that θ-subsume $H_\perp$, is combined via inference with the existing weighted CE pattern set H_t, yielding a concise set of CE patterns H_{new} , such that an optimal answer set of $B \cup H_t \cup H_{new} \cup I_t$ best-approximates the true state associated with I_t.
5. Each bottom rule $r_i \in H_\perp$ is "decomposed" in the following way. The head of r_i corresponds to an atom use$(i, 0)$ and each of its body literals, δ_i^j, to a try/3 atom, which, via the try/3 definitions provided, may be satisfied either by satisfying δ_i^j and an additional use(i, j) atom, or by "assuming" not use(i, j)
6. Choosing between these two options is done via ASP optimization

7. New rules are "assembled" from the bottom rules in $H_\perp$, by following the prescriptions encoded in the use/2 atoms of an optimal answer set of the resulting program

8. Once the new CE patterns are generated, their weights are updated based on their groundings in I_t and the true state. Moreover, each new pattern r is associated with the bottom rules from $H_\perp$, which are θ-subsumed by r_i. These bottom rules are used as a pool of literals for further specializing r over time, as described in the next section.

2.2.3.4 Updating the CE patterns' Structure

WOLED updates the structure of the existing patterns via a hill-climbing search process, by generating a bottom clause (rule) $\perp_\alpha$ from a CE instance α and then searches for a high-quality CE pattern into the subsumption lattice of $\perp_\alpha$ as shown in Figure 2.2. For the process of finding a high-quality CE pattern WOLED :

1. Starts from an empty body clause (parent rule) r' as at the top of Figure 2.2

2. At each time point t it evaluates a parent rule and its specializations on the incoming interpretation I_t via a scoring function G . Specializations are generated by adding a single literal to the parent rule r'. The scoring function G provides the gain, in terms of predictive performance on the inferred state, of a specialization rule r over the parent rule r'

3. The highest quality specialization $r1$ is found by a Hoeffding test [14].

4. Once the test succeeds, the parent rule r' is replaced by r

5. The process continues as long as new specializations of the current parent rule improve the rule's performance.

Algorithm 1: WOLED$(B, M, \mathcal{I})$

input: background knowledge B; mode declarations M; a stream of interpretations $\mathcal{I}$

$H_t := \emptyset$.
foreach *interpretation* $I_t \in \mathcal{I}$ **do**
  $I_t^{\text{MAP}} :=$ MAPInference(B, H_t, I_t).
  Receive I_t^{true}.
  Mistakes $:= I_t^{\text{true}} \setminus I_t^{\text{MAP}}$.
  $H_t \leftarrow$ SpecializePatterns(H_t).
  $H_{\text{new}} :=$ LearnNewPatterns$(B, M, I_t, I_t^{\text{MAP}}, I_t^{\text{true}})$.
  $H_{\text{new}} \leftarrow$ UpdateWeights$(H_t \cup H_{\text{new}}, Mistakes)$.
  $H_t \leftarrow H_t \cup H_{\text{new}}$.

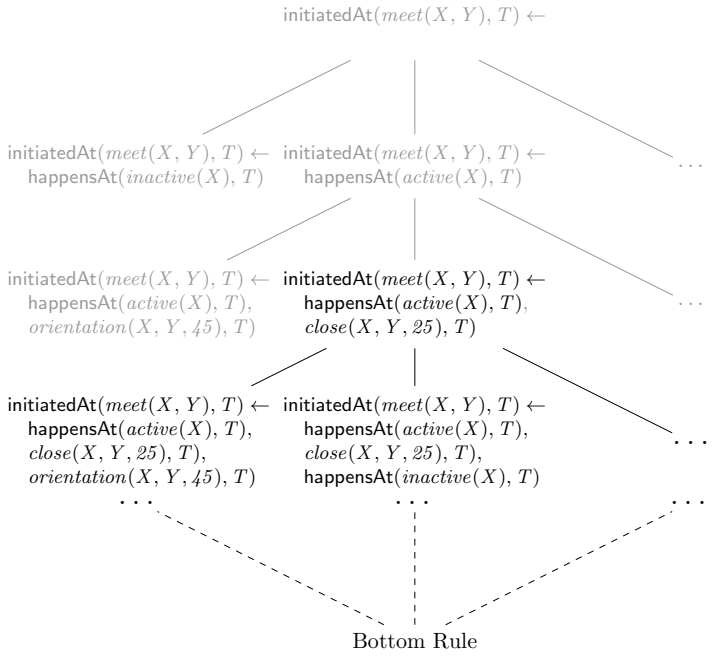

Figure 2.2: A subsumption lattice

2.3 Functional Geometric monitoring

The need to execute continuous queries over vast quantities of streaming data in distributed systems has been in focus in recent years' research. The Geometric Method, a communication protocol intended to be used in a distributed scenario for monitoring non-linear functions has been introduced in [28] and has proved to be an efficient solution for this rising problem. The appearance of the Geometric method, resulted in a series of research resulting in updates over the basic idea. Notably, the functional geometric monitoring (FGM) proposed in [1] was a substantial improvement over the Geometric Method. We provide the basic idea for the FGM as we used a monitoring approach for machine learning scenarios, proposed in [22], in the communication protocols implemented for WOLED as a distributed learning system.

2.3.1 Approximate query monitoring

Considering k remote compute nodes (sites) and a communication hub (coordinator). A local stream is either created or generated at each site. A stream can be denoted as a high dimensional vector $V = \mathbb{R}^D$. Each site updates V as stream updates arrive. Let $S_i(t), i = 1...k$ be the local state vector of each site i. All sites communicate with the communication hub where users can pose queries on the global data stream. We assume that at time t the global stream state is the average of the state vectors held at each site: $S(t) = \frac{1}{k}\sum_{i=1}^{k} S_i(t)$.

A continuous query $Q(S(t))$ is conventionally a non-linear function of the global stream

state S. In this context, communication can be significantly reduced between the coordinator and the sites by tolerating some small bounded error to the answer given to the user. At time t the coordinator possesses a close estimate of the global stream state $E(t)$ which can be used to give an approximate answer to a given query $Q(E(t))$, with a guarantee that for a given error ϵ the continuous query on the global stream Q$S(t)$):

$$Q(S(t)) \in (1 \pm \epsilon)Q(E(t)) \tag{2.4}$$

The sites periodically publish their local updated state vectors S_i to the coordinator. Sites respect the guarantee given for the bounded error by monitoring a local condition. If E_i denotes the local vector that was last sent by site i to the coordinator, and $E = \frac{1}{k}\sum_{i=1}^{k} E_i$, then no local site needs to publish its local updates as long as the true unknown global stream state $S(t)$ is within the admissible region

$$A = \{x \in V | Q(x) \in (1 \pm e)Q(E)\} \tag{2.5}$$

So the problem of approximately tracking the continuous query $Q(S(t))$ is analogous to monitoring the geometric condition $S(t) \in A$. Only when this condition is violated, the estimate E needs to be updated so as to restore the system invariant of equation 2.4.

2.3.2 Functional Geometric Monitoring for distributed data streams

Here we provide a basic explanation of the FGM protocol whose main purpose is to monitor the invariant of equation 2.4. This protocol, works in rounds where each round starts when a new estimate E is generated by the coordinator. At each time t a drift vector from the current estimate $X_i(t)$ is held at each site. The system is considered to be in a safe state as long as $\frac{1}{k}\sum_{i=1}^{k} X_i = S \in A$ where A is an admissible region.

A function $\phi : V \to \mathbb{R}$ is safe for an admissible region A, if for all $X_i \in V, i = 1...k$,

$$\sum_{i=1}^{k} \phi(X_i) \geq 0 \Rightarrow \frac{\sum_{i=1}^{k} X_i}{k} \in A$$

Given an admissible region A and a reference point E, a safe zone function ζ is a concave function which is safe for A and $\zeta(A) > 0$.

The following steps describe the FGM protocol which monitors the threshold condition $\sum_{i=1}^{k} \zeta(X_i) \geq 0$. This condition is equivalent to equation 2.4:

1. At the beginning of a round, the estimate E held by the coordinator is the true state of the system $E = S$. It selects a safe function ϕ. Let $\psi = \sum_{i=1}^{k} \zeta(X_i)$. The coordinator broadcasts ζ to all sites (the coordinator can also send E as A can be determined from it). The sites initialize their drift vectors to E. Thus at the start of each round $\psi = k\zeta(E)$.

2. After all sites have received the current estimate E a number of subrounds start. At the end of all subrounds $\psi > \epsilon_\psi k\zeta(E)$, for some small user specified ϵ_ψ and a new round should start.

3. The Coordinator ends the round by collecting the local drift vector of each site and updating the current estimate E.

For the subrounds, the FGM protocol needs to monitor that $\psi \leq 0$ with a precision of almost θ by engaging as less communication as possible:

1. At the beginning of a subround, the coordinator is aware of the value of ψ. It computes the subround's quantum $\theta = -\frac{\psi}{2k}$ which is shiped to all sites. Additionally the coordinator initializes a counter $c = 0$. Each local site records its initial value $z_i = \phi(X_i)$, where $2k\theta = -\sum_{i=0}^{k} z_i$. Finally each site initializes a local counter $c_i = 0$.

2. A drift vector X_i is maintnained at each site i. When X_i gets updated, site i updates its counter $c_i = max\{c_i, \lfloor \frac{\phi(X_i)-z_i}{\theta} \rfloor\}$. If the counter is increased, the site informs the coordinator for the increase to c_i.

3. When the coordinator receives an increment of a local counter c_i, it adds the increment to his global counter c. If $c > k$ the coordinator collects all $\zeta(X_i)$ from all sites and recomputes ψ. If $psi \geq \epsilon_\psi k\zeta(E)$, the current subround ends, or else another subround starts.

It is proven that as long as the global counter $c \leq k$ then $\sum_{i=1}^{k} \zeta(X_i) > 0$.

2.4 Related work

In recent years, a vast amount of research has been carried out on distributed systems. Communication protocols in distributed system (either static or dynamic), especially in the field of online machine learning, have been an important part of contemporary research. It is obvious that as the scale of such system grows with today's enormous amounts of data, effective communication protocols can ensure that performance will not be hindered.

Earlıer research on distributed online learning ([6], [29], [25], [7]) focused on static communication protocols. Such protocols update the global model after a fixed-size batch of data has been processed by the learners.

Michael Kamp proposed in [16], proposed the first dynamic model synchronization protocol for distributed online prediction that aimed further decrease the elevated communication cost in machine learning applications. Communication is engaged only in system states where the variance of the local models is high enough. This is accomplished by monitoring local conditions in each compute node. Further work based on this protocol was done in [17], [18], [15].

The Geometric Monitoring communication protocol for distributed systems was proposed in [28],provided a robust solution for monitoring complex queries over large scale distributed data streams using convex analysis theory. The Geometric monitoring protocol monitors a threshold function which is reduced to a set of local constraints monitored by the local sites. Later work ([10], [11], generalized the GM method by monitoring geometric constraints on distributed succinct summaries of streams, such as histograms or generally, high-dimensional vectors. Finally the Functional Geometric Monitoring proposed in [1] an improvement over the GM method, was adapted as a general communication protocol to be used in machine learning applications was proposed in [22].

3. DISTRIBUTED LEARNING APPROACH FOR WOLED

In this chapter, we perceive WOLED as a distributed learning system. We start deviating from the single learner approach that was described in 2.2. we examine how to extend the already existing learning process so as to be parallelized. First, for the execution of multiple learners, the actor model implementation of WOLED is presented. Moving on, for the distribution of the dataset we set up an Apache Kafka cluster. Finally the communication protocols between the learners are described in the next chapter as they represent the core of this work.

3.1 Distributed Machine Learning Systems

Distributed machine learning systems, use multiple compute nodes for the execution of a machine learning algorithm. The purpose of such systems is to increase the performance and accuracy of the single threaded version of an existing algorithm while also managing to deal with a much larger dataset [9]. Being able to handle large amounts of data can be equal in many cases to learning the given task more effectively. The steps for creating a distributed machine learning algorithm involve a mechanism to feed data to each node and the challenge of parallelizing the algorithm. This consists of establishing communication protocols that will enable the individual progress of each node to be shared among all nodes and averaging methods to combine the shared progress. In distributed machine learning systems one of two architectures can be followed: model parallelism and data parallelism. In model parallelism, each node has a part of the parameter vector of a neural network for example. In data parallelism, each node has the whole parameter vector and is responsible for a part of the dataset. The distinction is not strict as hybrid parallellism scenarios can exist.

3.1.1 Model based Distributed online learning scenario

Considering a network of k remote learners (compute nodes) and a Coordinator, In machine learning model based algorithms following the data parallelism paradigm, a copy of the model's parameters $w \in \mathbb{R}^D$ resides in each Learner, where D is the dimension of the parameter vector. Learners receive batches of data ,as an input, from the dataset and update their local copy of the parameter vector. When a learner updates the model's parameters, the updated parameter vector is delivered to the coordinator. This can be done in either a synchronous or an asynchronous way. The Coordinator is then responsible to gather the updates and integrate them into a global model via an averaging method. This scenario is presented as our communication protocols for WOLED's logic based machine learning will be executed in similar conditions that will be analyzed in the following sections.

3.1.2 WOLED as a distributed learning system

We approach learning using the same network topology where the network consists of k Learners and a Coordinator. First we take a look into the data distribution mechanism. The Caviar dataset, consists of a number of videos in the form of first order logic narrative as in 2.1(c). The videos are distributed across the learners in a round robin fashion via a Kafka topic with k partitions. As for the parellization, a core difference from a classic model based Machine Learning algorithm, as described above, is the absence of a fixed size parameter vector. The equivalent of the parameter vector of a Learner i in a given time point t is the theory $H_i t$ consisting of weighted initiatedAt/2 and terminatedAt/2 CE patterns as described in 2.2. However in each t_j after t, it is possible that new rules are added and so the size of the theory could be extended. The coordinator needs to not only average the weights of already existing rules, but to also integrate newly created ones to the new global theory and thus this approach does not clearly fall in the category of either data parallelism or model parallelism. The communication between the learners and the coordinator is accomplished using the actor model as described further below. In the next chapter, we try and evaluate different methods concerning the communication and merging between the learners and the coordinator.

3.1.3 The Actor Model

To provide a way for learners to share their updated CE pattern theory with the coordinator, the actor model has been used. The actor model is a is a mathematical model of concurrent computation that was introduced by [13]. Actors, the fundamental computation unit, interact with each other by exchanging messages. They are unable to intervene directly to the other actors' execution flow and can only alter their own private state. The only way an actor can influence the functionality of other actors is by sending a messages. This model, can be utterly beneficial in modern applications especially for the development of distributed systems and consequently distributed machine learning. By encapsulating the state of each actor and eliminating the need for shared memory, one can focus on the development of the system rather than low-level protocols.

The means for actors to exchange messages are each actor's private mailbox. Actors knowing the address of a mailbox, can directly send different kinds of messages to it. An actor can process messages he receives in its mailbox in a FIFO order (by default). A behavior is set to control how and what kinds of messages will be processed. Upon receiving a message an actor can take action and manipulate its current state by depending on what kind of message was delivered. The most common actions an actor can perform are:

1. Send messages to other known actors . Each actor has a unique id and can be identified by other actors that are aware of this id.

2. Execute other actors. A parent child relationship is established whenever an actor

starts another actor. In this case, child actors can automatically identify their parent and can easily communicate.

3. Choose a different behavior for handling following messages. A behavior, is a loop function that repeatedly checks for messages in the actor's mailbox and is responsible for what messages will be processed and actions will be taken upon receiving new messages.

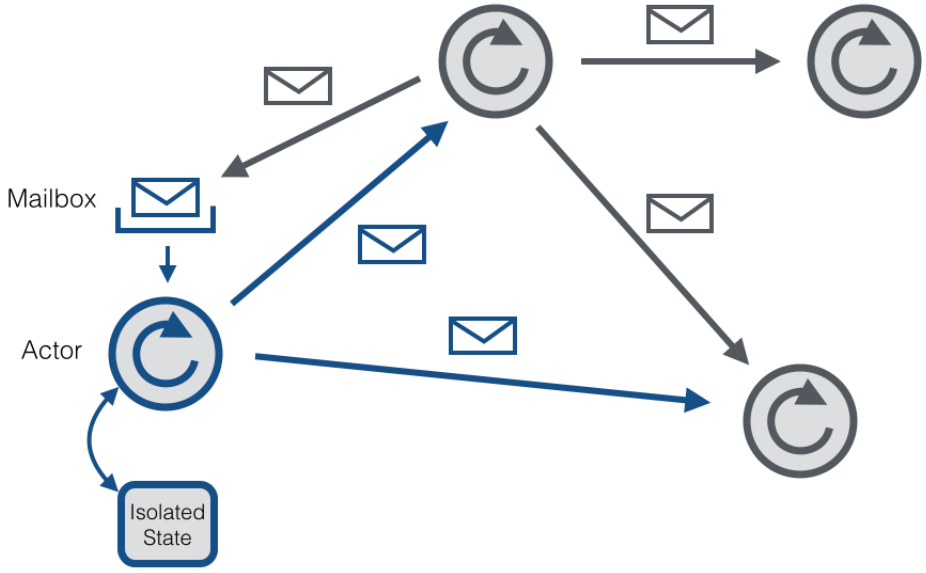

Figure 3.1: An overview of the actor model

3.1.4 WOLED's actor based implementation

To implement the distributed learning network consisting of the learners and the coordinator, the actor model proved to be a convenient solution as the messaging system can cover the communication needs between the learners and the coordinator, while each process can run individually without the need for resource access locks. Having a resilient communication system established, we can focus on protocols oriented to how and when communication will happen(i.e when should a learner send updates of its local theory and how the coordinator will merge the local theories and distribute back the updated theory.) The structure of this actor based implementation is as follows:

- The Coordinator is the first actor to be executed. In the beginning the coordinator forks k child actors representing the learners.
- By sending an initiation message to each learner, the coordinator signals the learners to start processing batches. In the meantime the coordinator waits for update messages from the learners.

- Depending on the communication protocol applied in each case when enough updated have been made to a learner's local theory, a message containing weight updates of the rules and/or newly derived rules is sent to the parent (the coordinator). The learners then wait for a message containing the updated theory.
- The Coordinator averages the weights of existing rules, integrated the new rules and assembles a global theory that is then distributed to all learners.
- When the learners receive the merged theory the learning process continues.

At this point, it is worth mentioning that for theory sharing, learners and the coordinator do not broadcast entire clause objects for each Complex Event pattern in the theory. A string representation of each pattern accompanied by the pattern's statistics (TPs, TNs, FPs, FNs) is contained in each message with respect to avoiding unneeded communication cost.

3.1.5 Our Data parallelism approach with Kafka

To split the dataset to multiple learners, Apache Kafka is a fault-tolerant high throughput publish-subscribe based messaging system. In Kafka terminology a topic is a "category" where messages (called records) can be sent. Each topic can be split into multiple partitions. Producers are applications that can send messages to a specific topic by subscribing to it. Consumers are the exact opposite. They are applications that can read messages from the topic. There are multiple ways in which the whole process of message exchange can be configured. In our case, we used a topic consisting of k partitions (one partition per learner). The WOLED learners are consumers subscribed to the topic, where each one is assigned a partition. All messages in a specific partition are consumed by the same learner. As mentioned above, The CAVIAR dataset is a collection of videos. To equally distribute the videos to the learners each video is split to mini-batches (Interpretations) and sent as records to the topic in a round robin fashion. In this way all batches created from the same video are processed by the same learner. Considering future updates, Apache Kafka is a fitting platform as it can easily scale to handle a substantial amount of data and simultaneous producers/consumers.

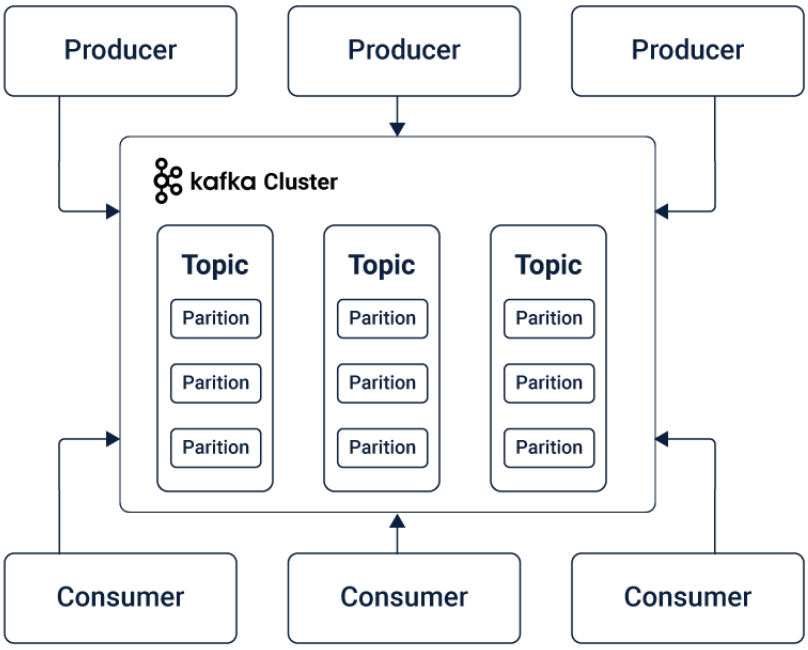

Figure 3.2: Example of an Apache Kafka cluster with three topics, each having three partitions. In our case we use producer applications to distribute the dataset between topics. The learners are conxumer applications that can then individually read input data from the topics at their own pace.

4. COMMUNICATION PROTOCOLS

Our main goal is to apply different possible protocols for the coordinator to collect and merge the local theories during the learning process in order to reach a satisfactory level of predictive performance in the fastest way while also minimizing the communication cost. All methods follow the distributed learning scenario described in 3.1. All communication protocols follow the same basic principle:

- At each time point, the coordinator holds a global theory consisting of CE patterns in the form of first order logic rules.
- The learners start updating their local copy of the global theory until, depending on the protocol applied, the local updates are sent to the coordinator
- The coordinator merges the local updates and integrates them to the global theory.
- The new global theory is sent back to all learners and the learning process is resumed in the same way.

We compare four different communication protocols taking into consideration the communication cost and prediction error against the standard single Learner implementation. The results of the three methods as well as their execution environment are described in the next chapter.

4.1 Remote learning without communication

This is a trivial method in which the Coordinator starts k Learners that do not communicate during the learning process. Each learner receives and processes batches of Interpretations from the Kafka topic, thus resulting in each learner coming up with a different theory. A method having no significant accuracy gains compared to this method could be considered unavailing as the added communication cost can outweigh the overall gain. Thus this method is used to set a base performance as a comparison for the two following ones.

4.2 Synchronous theory merging after a fixed number of batches

This is a simple merging method in which the coordinator synchronously gathers the updates after a predefined number of batches have been processed. Following the same distributed scenario depicted at 3.1, at the beginning of the execution, the coordinator starts k Learners. Each WOLED learner, starts processing batches of simple event observations, from a Kafka topic. After having processed m batches, each learner, sends its local theory updates to the coordinator and waits for a response message with the global merged theory containing averaged updates accumulated by all learners. Formally:

- The coordinator holds at step s the global theory H_s containing the rules, their respective weights and statistics (TPs, TNs, FPs, FNs) for each rule:

$$H_s = r_1 w_1, r_2 w_2, ..., r_n w_n$$

- H_s is sent to the learners. By processing batches, a learner updates the weights and statistics of the already existing rules and possibly adds l new rules. After having processed m batches, the theory at learner i after step s is:

$$H_s(i) = r_1 w'_1, r_2 w'_2, ...r_n w'_n, r_{n+1} w_{n+1}, ..., r_l w_l$$

- Each learner i sends his updated local theory $H_s(i)$ to the coordinator.
- After all learners have sent $H_s(I)$ to the coordinator, the new global theory H_{s+1} is generated by:
  1. Averaging the updates in each H_s (i) applied to the weight of each rule in H_s: $w_{j(s+1)} = \frac{\sum_{i=1}^{k} w_j(i)}{k}$
  2. Adding up new predictive statistics i.e. if in H_s rule r_1 had $20\ TPs$ and after the predictions of learner i during the processing of the next m batches, in $H_s(i)$ rule r_1 has $40\ TPs$, in H_{s+1} rule $r1$ will have $40\ TPs$
  3. append the l newly generated rules of every learner

4.3 Continuous theory averaging after every batch

This communication protocol differs to the above described Synchronous theory merging after N batches only to the fact that each learner i sends its local theory $H_t(i)$ to the coordinator after processing every batch. This protocol is included to the experiments conducted as it can provide an overview of the trade-off between communication cost and training loss. The later we engage communication the less messages are transmitted but new CE patterns generated by a learner are not getting across the rest of the learners until the synchronization. The goal is to minimize the communication cost while also keeping the accuracy close to the single learner scenario.

4.4 Functional Geometric Monitoring (FGM) protocol for learning

In 2.3 we briefly described the Functional Geometric Monitoring protocol that was intended for monitoring complex queries over distributed data streams. An adaptation of this method for distributed machine learning was introduced in [22]. Using the distributed learning implementation described in 3.1, this framework allows for distributed online training of Machine Learning algorithms by utilizing the functional geometric monitoring protocol. The coordinator holds the global model which is broadcast to all sites. Sites update

their local copy of the model as data arrives as an input. By having each site monitor a local condition, communication is engaged only when it deems necessary and thus the communication cost of the learning process is minimized.

The concept behind this method is to have the coordinator collect updates and use an averaging method only when the copies of the models residing in each learner have drifted enough from the global estimate E. The global estimate E is the parameter vector of a given training algorithm. This protocol works in rounds and subrounds similarly to the FGM protocol for distributed data streams. For a given safe zone function ζ, as long as $\sum_{i=1}^{k} \zeta(X_i) > 0$,where X_i is the drift vector (i.e the updated parameter vector) held at learner i, no communication is necessary and the round continues. The above global condition is decomposed into a set of local conditions to be monitored by each individual learner. It is proven that if the local conditions hold, the global condition holds as well and the system is still within the safe zone.

When the global condition is violated, the local model parameter vectors are collected by the coordinator, an averaging method is applied and the resulting global model is used as the new estimate E'. The new estimate is sent back to all learners replacing their local model and a new round starts.

A rebalancing method has also been proposed in the aforementioned work, but here we solely focused on the basic version of the FGM protocol for machine learning as the adaptation of WOLED as a distributed system is still in an experimental state. As a first step the goal would be to adapt the existing learning process to simple communication protocols that will bring forth the changes that need to be made to the existing algorithm before moving on to more complex ones.

Algorithm 2. shows the adaptation of the functional geometric protocol for machine learning.

4.4.1 WOLED's adaptation to the FGM online learning protocol

We tried to apply an adaptation of this method to WOLED's learning process using the variance of the model as a safe zone function $\zeta(X_i) = \sqrt{T} - \|X_i - E\|$ where T is a user given threshold, X_i the weight vector of the theory held at Learner i and E The round's theory weight vector estimate. The main difference to a common machine learning scenario, is that while a model has a fixed size parameter vector $w \in \mathbb{R}^D$, WOLED starts with an empty theory (the initial weight vector is of size 0) and each learner adds newly generated rules while learning is in progress. To address this issue, we firstly use enough batches in the warm up round to produce an adequate number of CE patterns in the initial theory.A smaller warm up dataset, would mean more communication engaged on the first rounds of the protocol's execution. In addition to that, for the purpose of making the computation of $\|X_i - E\|$ possible in each learner, we expand the estimate with zero valued weights for every newly created rule. In more detail:

- At time t the estimate E consists of k weights of rules, denoted $w1, w2, .., wk$.

- At time $t+1$, l rules are added to a learner's theory H_{t+1} and possibly some of the already existing rules' weights are altered. So H_{t+1} contains: $w'_1 r_1, w'_2 r_2, ..w'_k r_k, w_{k+1} r_k .. w_l r_l$. The current X_i is $w1', w2', ..wk', w_{k+1}, .., w_l$.
- Clearly the estimate has k elements, whereas X_i has $k + l$ elements. In order to compute the difference $X_i - E$ we extend E by adding l zeroes to the end.

One issue that arises with this logic based approach, is the absence of the concept of convergence. New batches of simple event observations in the learning dataset can alter the weights of existing rules of a learner's theory significantly. Even though most occurrences of the target complex follow similar patterns, variations can be observed. This is reflected to the learning process by the promotion or demotion of different rules that performed better while generating the inferred state in such cases. Thus the complex event patterns (rules) that form the existing theory can move to a different point. So considering the nature of Complex Event Recondition, the condition indicating the safety of the system can be often violated, resulting in growth of the communication cost . A trade-off that can be made to reduce communication cost is to give the system a larger threshold T for the safe zone function $\zeta(X_i) = \sqrt{T} - \|X_i - E\|$.

Algorithm 2: ML-FGM

Initialization *at the coordinator*:

Warm Up the global learner and end up with parameters w_{init}
Set containing the Nodes that have updated their loca parameters: $U \leftarrow \emptyset$
$E \leftarrow, c \leftarrow 0, \psi \leftarrow k\zeta(E), \theta \leftarrow \frac{\psi}{2k}$
send E and θ to all learners and start the first round

A. *Site i on **receiving** E and θ at the start of a new round*:

update the local model: $X_i \leftarrow E$
$quantum \leftarrow \theta$, $c_i \leftarrow 0$, $z_i \leftarrow \zeta(E)$

B. *Site i on **receiving** θ at the start of a new subround*:

$c_i \leftarrow 0$, $quantum \leftarrow \theta$, $z_i \leftarrow \zeta(X_i)$

C. *Site i on **observing** a batch at time t*:

update the local model X_i by fitting to it $batch_i$
$BatchesObserved_i \leftarrow BatchesObserved_i + 1$
if $BatchesObserved_i$ *mod* $m = 0$ ***and*** $\lfloor \frac{z_i - \zeta(X_i)}{quantum} \rfloor > c_i$ **then**
  $Increment_i \leftarrow \lfloor \frac{z_i - \zeta(X_i)}{quantum} \rfloor - c_i$
  $c_i \leftarrow \lfloor \frac{z_i - \zeta(X_i)}{quantum} \rfloor$
  send $Increment_i$ to the coordinator

D. *Coordinator on **receiving** an increment*:

$c \leftarrow Increment_i$
if $c > k$ **then**
  request and **collect** all $\zeta(X_i)$ from all sites
  $\psi \leftarrow \sum_{i=1}^{k} \zeta(X_i)$
  if $\psi \leq \epsilon_\psi k\zeta(E)$ **then**
    request and **collect** all ΔX_i from all sites
    for *each* ΔX_i **do**
      if $\Delta X_i \neq 0$ ***and*** $i \notin U$ **then**
        augment U **with the site** i
      $E \leftarrow E \frac{1}{size(U)} \sum_{i=1}^{k} \Delta X_i$
      $U \leftarrow \emptyset$, $c \leftarrow 0$, $\psi \leftarrow k\zeta(E)$, $\theta \leftarrow \frac{\psi}{2k}$
      send E and θ to all sites and start a new round (code A)
  else
    $c \leftarrow 0$, $\theta \leftarrow \frac{\psi}{2k}$
    send θ to all sites to start a new subround (code B)

5. EXPERIMENTAL EVALUATION

We now present the results gathered from applying the communication protocols described in chapter 4 to the actor based implementation of WOLED described in 3.1.4, along with the setting (execution environment, dataset) in which the experiments were conducted.

5.1 Execution environment

5.1.1 Goal

In order to evaluate the efficiency of the four communication protocols that were implemented, namely: No Communication, Continuous Communication, Fixed synchronization and FGM Dynamic synchronization, we used the Complex Event Recognition task of constructing a set of Complex Event patterns (theory). The purpose of the theory is to detect occurrences in the input data of 1) two people meeting and 2) two people moving together. In each execution a target Complex Event is chosen and patterns for recognizing this specific complex event are generated.

5.1.2 Experiment structure

As mentioned in 2.1.3 we used the CAVIAR dataset as input in our tests. The CAVIAR dataset consists of videos taken in a public space, where actors perform a given activity. The CAVIAR team has provided annotations for these videos in the form of observations of activities in each time point. These observations are in the form of first order logic facts. In our case we used the part of the dataset containing the descriptions of the videos showing people meeting and moving together.

To better evaluate the learning performance of each different online training scenario 10 fold cross validation was applied. 90% of the dataset used as a training set and 10% of the dataset used as a testing set The training set is used for generating CE patterns and update the weight of already existing ones, while the testing set is used to assess the performance of the CE patterns.

The dataset was repeated five times to adequately distribute data to all learners. As each learner processes batches of simple event observations, a batch size of 50 was used, i.e an interpretation I_t containing 50 simple events was processed at each given time t.

We compare the four communication protocols applied on the distributed implementation of WOLED with the monolithic implementation of WOLED (mentioned as Single Core in the results) as described in 2.2. We split the dataset to 2, 4, 8 and 16 learners to incrementally observe the performance in execution time and learning process as well as the communication cost.

5.1.3 Computing environment

All experiments were conducted on the same 3.6GHz processor (4 cores, 8 threads). In this context it is worth mentioning that even though we did not use remote machines, with the use of actors the distributed learning scenario can be simulated as in a real world application actors could reside in different machines. Communication cost can still be measured by the messages exchanged by the actors and has an impact on training times as each time communication is engaged theory merging and thus theta-subsumption between the CE patterns (rules) is needed. The only difference is that network delay is not taken into account for the execution times.

5.1.4 Result structure

The reported values in Table 5.1 and Table 5.2 are the averages measured from the 10-fold cross validation process mentioned above. To compare the performance of our Distributed learning approach we evaluated the aforementioned communication protocols on:

- The communication cost between the learners and the coordinator measured in KBs. The communication cost consists of the messages exchanged by the coordinator and the learners containing the Complex Event patterns. This cost is be greatly reduced by sending a string representation of the patterns accompanied by their statistics, instead of the object used during the learning process.
- The speedup in execution time of the distributed learning approach following the different communication protocols against the Single-core implementation:

$$speedup = \frac{SingleCoreTime}{DistributedTime}$$

- The F1-score that was achieved by predicting occurences of the given CE (meeting or moving) in the test set. The F1-score is the harmonic mean of the precision and the recall:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = \frac{TPs}{TPs + \frac{1}{2}(FPs + FNs)}$$

  (WHY IS IT USED?)

- The average prequential loss , which refers to the average online error which is computed as:

$$AverageLoss = \frac{TotalMistakes}{NumberOfBatches}$$

5.2 Results for Meeting and Moving

The accumulated results are shown in the two tables below.

Table 5.1: Meeting Results

	Learners	Time(sec)	Speed-up	F_1-score (test set)	Preq. Loss	Comm. Cost
Single Core	1	1398	–	7.18	0.848	–
No Communication	2	667	2.09	9.23	0.848	–
	4	385	3.63	8.78	0.848	–
	8	278	5.02	9.18	0.803	–
	16	224	6.24	9.85	0.824	–
Cont. Communication	2	982	1.42	8.12	0.824	152
	4	612	2.28	7.56	0.803	423
	8	443	3.15	7.87	0.834	945
	16	318	4.39	8.78	0.824	2134
Fixed-sync (every 10 batches)	2	723	1.93	9.13	0.823	134
	4	421	3.32	8.92	0.823	278
	8	313	4.46	9.95	0.799	506
	16	296	4.72	10.48	0.808	1134
FGM dynamic-sync	2	575	2.43	7.98	0.838	54
	4	312	4.48	8.14	0.838	123
	8	223	6.26	8.34	0.824	312
	16	168	8.32	8.65	0.8	423

Table 5.2: Moving Results

	Learners	Time(sec)	Speed-up	F_1-score (test set)	Preq. Loss	Comm. Cost
Single Core	1	1645	–	9.24	0.803	–
No Communication	2	785	2.09	11.24	0.812	–
	4	448	3.67	11.78	0.798	–
	8	378	4.35	13.14	0.792	–
	16	213	7.72	12.87	0.801	–
Cont. Communication	2	1134	1.45	9.84	0.767	213
	4	978	1.68	10.12	0.798	508
	8	645	2.55	9.65	0.801	1323
	16	512	3.21	11.35	0.791	3523
Fixed-sync (every 10 batches)	2	825	1.99	10.86	0.768	152
	4	482	3.41	11.35	0.768	321
	8	412	3.99	12.54	0.778	718
	16	357	4.6	12.67	0.763	1478
FGM dynamic-sync	2	812	2.02	10.78	0.802	82
	4	421	3.09	10.24	0.768	176
	8	318	5.1	10.3	0.768	323
	16	265	6.2	11.28	0.76	533

6. CONCLUSIONS AND FUTURE WORK

At this point, we present a summary of the work that has been done in the course of writing this thesis along with our conclusions on the results presented in the previous chapter. Finally we recommend future work to tackle the problems and limitations we came across as well as to enhance the results.

6.1 Synopsis

The purpose of this work was to expand WOLED functionality by approaching it in a distributed learning scenario. We started from an actor based approach of the algorithm consisting of the learner actors and the coordinator actor. Different communication protocols were applied between the learners and the coordinator in order to share the individual learning progress of each learner. Our main goal was to minimize the communication cost and training time while keeping the training loss as close as possible to the single learner approach. We tried four different protocols and utilized both static and dynamic averaging. From the implementations that were tested our interest was directed towards weather the dynamic FGM protocol would outperform the fixed synchronization methods or vice-versa.

6.2 Result Conclusions

It is evident from observing the results in Table 5.1 and Table 5.2 that even though no communication protocol was found to scale up the training performance (F1- score and prequential loss) significantly, The FGM dynamic synchronization method was able to both keep a score close to the single learner approach and minimize the communication cost and training time.

In more detail, all distributed learning approaches do not show big differences in F1-score and prequential loss compared to the single core approach. This happens due to the fact that the data distributed between the learners are almost identical and thus similar Complex Event patterns are produced. However this is a limitation of the dataset and is probably something that can be addressed in the future. The continuous communication was significantly slower and evidently had the biggest communication cost. This can also be seen as an adequate measure of the time needed for the syncing of the theories. Fixed Synchronization, as expected, was faster and required far less communication than the continuous communication. Finally the FGM dynamic synchronization was proved to be the best communication protocol both in terms of speedup and communication cost while the training performance stayed closer to the Single Core than the Fixed Synchronization. The results we got from the FGM prove that dynamic synchronization can be utterly beneficial for WOLED in a distributed learning scenario and gives the ability for scaling the experiments to even larger input data and more learners working cooperatively.

6.3 Future Work

As WOLED is a relatively new concept and is still in an experimental state, there is still a lot to be expected in terms of future extensions. In this work, we proved that WOLED can be extended to effectively work as a distributed learning system. However in this work we only took the first steps needed to adapt this algorithm to real world circumstances.

It is clear from the experimental evaluation, that WOLED as a distributed learning system can benefit from the utilization of a dynamic communication protocol such as the FGM protocol for learning. To this end, as the FGM was our first attempt at a dynamic protocol, more similar protocols could be applied to WOLED actor based implementation. In addition to that as the FGM protocol involves the selection of a safe zone function which is an important part of the algorithm's performance, we could study and apply different functions that would better fit WOLED's theory weight vector update scheme.

As we only executed our test in a single machine by simulating a multi-node scenario, in future work we ought to experiment with executing actors in different machines, as in a real world deployment scenario. By doing so we could evaluate the added network cost as well as find robust ways to deal with system failures.

Finally, a limitation of the data set for our current work was found. As the dataset was relatively small the repetition of the same data among the learners in order to have a substantial amount of training data was inevitable. By providing a larger and more diverse dataset we could evaluate with better accuracy the performance of different communication protocols.

ABBREVIATIONS - ACRONYMS

CER	Complex Event Recognition
CE	Complex Event
EC	Event Calculus
GM	Geometric Monitoring
FGM	Functional Geometric Monitoring
ML	Machine Learning

BIBLIOGRAPHY

[1] *Functional Geometric Monitoring for Distributed Streams*. Zenodo, March 2019.

[2] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and George Paliouras. Probabilistic complex event recognition: A survey. *ACM Computing Surveys*, 50, 02 2017.

[3] Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27(4):469–506, 2012.

[4] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3), June 2012.

[5] Luc De Raedt. *Logical and relational learning*. Springer Science & Business Media, 2008.

[6] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches, 2010.

[7] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 713–720, 2011.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[9] Alex Galakatos, Andrew Crotty, and Tim Kraska. *Distributed Machine Learning*, pages 1196–1201. Springer New York, New York, NY, 2018.

[10] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proc. VLDB Endow.*, 6(10):937–948, August 2013.

[11] Minos N. Garofalakis and Vasilis Samoladas. Distributed query monitoring through convex analysis: Towards composable safe zones. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[13] Carl Hewitt, Peter Boehler Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *IJCAI*, 1973.

[14] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 97–106, New York, NY, USA, 2001. Association for Computing Machinery.

[15] Michael Kamp, Linara Adilova, Joachim Sicking, Fabian Hüger, Peter Schlicht, Tim Wirtz, and Stefan Wrobel. Efficient decentralized deep learning by dynamic model averaging. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 393–409. Springer, 2018.

[16] Michael Kamp, Mario Boley, Daniel Keren, Assaf Schuster, and Izchak Sharfman. Communication-efficient distributed online prediction by dynamic model synchronization. volume 1018, 09 2014.

[17] Michael Kamp, Mario Boley, Michael Mock, Daniel Keren, Assaf Schuster, and Izchak Sharfman. Adaptive communication bounds for distributed online learning. 01 2014.

[18] Michael Kamp, Sebastian Bothe, Mario Boley, and Michael Mock. Communication-efficient distributed online learning with kernels, 11 2019.

[19] Nikos Katzouris. WOLED: A Tool for Online Learning Weighted Answer Set Rules for Temporal Reasoning Under Uncertainty. Zenodo, September 2020.

[20] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.

[21] NIKOS KATZOURIS, ALEXANDER ARTIKIS, and GEORGIOS PALIOURAS. Online learning of event definitions. *Theory and Practice of Logic Programming*, 16(5-6):817–833, 2016.

[22] V. B. Konidaris. Distributed machine learning algorithms via geometric monitoring, bsc thesis, 2019.

[23] Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer, 1989.

[24] Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR*, pages 145–154. AAAI Press, 2016.

[25] Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Daniel Walker IV. Efficient large-scale distributed training of conditional maximum entropy models. In *Neural Information Processing Systems (NIPS)*, 2009.

[26] Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009.

[27] Murray Shanahan. The event calculus explained. *In Artificial Intelligence LNAI*, 1600, 06 2000.

[28] Izchak Sharfman, Assaf Schuster, and Daniel Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Trans. Database Syst.*, 32, 11 2007.

[29] F. Yan, S. Sundaram, S. V. N. Vishwanathan, and Y. Qi. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2483–2493, 2013.