Knowledge extraction from the learning of sequences in a long short term memory (LSTM) architecture

Ikram Chraibi Kaadoud[12] Nicolas P. Rougier[123] Frédéric Alexandre[123*]

[1]Inria Bordeaux Sud-Ouest, 200 avenue de la vieille tour, 33405 Talence, France
[2]LaBRI, UMR 5800, CNRS, Université de Bordeaux, Talence, France
[3] Institut des Maladies Neurodégénératives, UMR 5293, CNRS, Université de Bordeaux, Bordeaux, France
[*] Corresponding author: {frederic.alexandre@inria.fr}

Abstract

Transparency and trust in machine learning algorithms have been deemed to be fundamental and yet, from a practical point of view, they remain difficult to implement. Particularly, explainability and interpretability are certainly among the most difficult capabilities to be addressed and imply to be able to understand a decision in terms of simple cues and rules. In this article, we address this specific problem in the context of sequence learning by recurrent neuronal models (and more specifically Long Short Term Memory model). We introduce a general method to extract knowledge from the latent space based on the clustering of the internal states. From these hidden states, we explain how to build and validate an automaton that corresponds to the underlying (unknown) grammar, and allows to predict if a given sequence is valid or not. Finally, we show that it is possible for such complex recurrent model, to extract the knowledge that is implicitly encoded in the sequences and we report a high rate of recognition of the sequences extracted from the original grammar. This method is illustrated on artificial grammars (Reber grammar variants) as well as on a real use-case in the electrical domain, whose underlying grammar is unknown.

Keywords: Sequence learning, Knowledge extraction, Long Short Term Memory, Latent space, Implicit learning, Recurrent Neural Networks

1 Introduction

Efficient numerical models are exploited as standard tools to process data in many domains of the socio-economic world. The possibility to extract knowledge from this kind of models becomes an increasingly important demand, as it is frequently discussed in the context of ANNs (Artificial Neural Networks). It is a valuable asset when one wants to compare such models with more traditional and explicit algorithmic approaches. It is even more fundamental, if not mandatory, in critical domains where the functioning of an autonomous decision-making system must be assessed by humans, in order to check that it relies on valid cues. Transparency is often evoked to justify this requirement (Lipton, 2016), advocating that the end-user of the model should have a level of understanding comparable to that of the expert, designer of the model and owner of the hidden knowledge. This is also true when the underlying knowledge is not known a priori and when the assessment is not a simple verification but rather corresponds to the discovery of a hidden knowledge that might be confirmed a posteriori. This situation can be compared, in human cognition, to the distinction between explicit and implicit memory (Reber,

2013). In the case of explicit memory, the knowledge is conscious and human decisions can be explained and expressed in a declarative way (you can for example explain that you have ranked students by comparing their marks following explicit rules). In the implicit case, you can acquire skills without being conscious of what you have learned and without being able to explain it. Consider, for example, typing on a keyboard a series of symbols where repeated sequences are hidden (Pascual-Leone et al., 1995). You improve your motor skills of reaching keys without being conscious of the existence of regularities. Language is another typical example of implicit learning acquired by children through practice (repetition and imitation), not through explicit learning of valid rules (Oudeyer, 2006). Implicit learning is pervasive in the socio-economic world because experts in a domain tend to acquire their expertise through practice and are, most of the time, unable (or only partly able) to verbalize their knowledge. Such expertise can also be acquired by models like ANNs by learning from examples produced by experts in order to reproduce the corresponding skill. In this case, the importance for knowledge extraction is high because this knowledge is hidden to all the actors of the process and might bring valuable information. As reviewed below, many approaches for knowledge extraction using ANNs have been proposed. In this paper, we will consider the process of knowledge extraction from the implicit learning of temporal behaviour. This means that the numerical models to consider will have to be able to process sequences (as it is the case with Recurrent Neural Networks (RNNs) for example) and the data to be manipulated will be consequently corpuses of sequences. Preliminary attempts for extracting knowledge from the Simple Recurrent Network (SRN) model have already been proposed (Elman, 1990; Servan-Schreiber et al., 1988; Cleeremans et al., 1989; Servan-Schreiber et al., 1991; Cleeremans and McClelland, 1991). But it was established that due to the limited size of the context layer, the SRN couldn't handle long term dependencies : the old past is overwritten by the recent past, which makes the development of a stable implicit representation of grammar rules impossible, especially in the case of long and ambiguous sequences. In the present paper, we propose to consider more complex and powerful RNNs, namely LSTM (Long Short Term Memory) networks (Hochreiter and Schmidhuber, 1997), since that model showed great performance for learning sequences from different kinds of dataset (Greff et al., 2017). Before diving into the methodology, we will first discuss some important aspects of the scientific context of knowledge extraction in machine learning (section 1.1), introduce experimental contexts that are classically considered in implicit learning of sequences (section 1.2), describe a classical process of knowledge extraction from RNNs (section 1.3) and report the current level of performance of this process with different kinds of RNNs (section 1.3.3), thus motivating our positioning (section 1.4).

1.1 Interpretability in machine learning

The principles of learning algorithms associated to multi-layered supervised neuronal architectures are known for more than thirty years (Rumelhart and MacClelland, 1986). These architectures have recently gained a renewed interest mainly because larger corpuses and the corresponding computing power are now available. This led to the development of innovative architectures such as for example convolutional networks in the domain of image classification (Goodfellow et al., 2016) or Long Short Term Memory (LSTM) models for temporal sequence processing (Hochreiter and Schmidhuber, 1997). These latter models have outperformed simple recurrent networks (SRN) in sequential data processing and are able to deal with complex structured sequences. However, and in spite of these major improvements, a major weakness remains for these multi-layered architectures: They are considered as black boxes and their explainability, or put differently their capacity to explain how they build their decision in a way intelligible for humans, is very weak or even non existent.

Several notions have been proposed to describe the processes of knowledge extraction from

these black boxes (Guidotti et al., 2018; Ayache et al., 2018). While explainability is the ultimate step where the functioning of the black box is fully understood and can be transferred to humans in understandable terms, interpretability refers to the capacity of breaking down all the inner mechanisms of the black box (without necessarily understanding them). This might be useful for an expert but not always for the layman. Interpretability is considered "global" when its purpose is to explain the whole logic of the model with rules associating the output with the input variables (for example approximating a non linear black box to a simple linear model). It is defined "local" when the process focuses on understanding specific reasons for a decision on a datum by extracting patterns critical to the decision (for example extracting a surface of separation between categories). These examples also indicate that interpretability can be achieved by an approximation process and can give an intuition rather than the full knowledge (Guidotti et al., 2018). These concepts (*i.e.* explainability and interpretability) are often mixed up in the literature, even if they are distinct since an interpretable model is a requirement for having an explanation which in turn depends, among other things, on the intended purpose, the decision-maker, and the context of the situation. In this paper, we will remain on the technical side and focus on the concept of interpretability and its dimensions.

In this specific field, Remm and Alexandre (2002) have defined two approaches for extracting knowledge from neural networks, seen as black boxes : the pedagogical method when rules can be extracted directly during learning, and the decompositional method, when elements of the rules are extracted from the hidden layers after learning before being recombined globally (in this case, we evoke a post-hoc interpretability).

Concerning RNNs, in the pedagogical case, some methods have proposed, querying an RNN using examples and counter-examples. The resulting pairs of (input, output) are used to confront the network against various hypotheses and directly model its global behaviour (Weiss et al., 2017) without considering what happens inside.

In the decompositional case, and specifically in post-hoc interpretability, knowledge is extracted a posteriori from the hidden layers of the network (Lipton, 2016). Indeed, it is admitted that during the learning phase, the network will extract automatically simple hints or features (knowledge of features) or privileged relations expressing simple causal relations between the input and the output spaces (knowledge of rules) that are relevant to the targeted task. This knowledge representing the implicit representation of the network is encoded into its latent space (Abu-Mostafa, 1990; Bengio et al., 2013) which is "hidden-layer-dependant". So by extracting and analyzing the activity of the hidden layers during the test phase (once the learning is finished), it is possible to get hints at interpreting the network behavior and its decisions.

Decompositional approaches have been proposed in the case of pattern recognition (Setiono and Liu, 1996; Remm and Alexandre, 2002). They include a step of pruning of the network, for the hidden layer to remain in tractable dimension, and another step of simplification by mapping the activity of the hidden units onto a finite set of values, corresponding to the features to be extracted. This allows to have a discrete and limited set of combinations to consider, to link input states to the resulting activation of the output, thus yielding rules. This approach has led to sound feature and rule extractions in several application domains. However, when dealing with recurrent networks processing sequences, the extraction task becomes much more difficult.

1.2 Implicit learning of sequences

The form of knowledge acquired by learning is a major question in cognitive science and several experimental protocols have been designed to study various paradigms in the case of implicit learning of sequences (Perruchet and Pacton, 2006; Pothos, 2007). There are two major approaches used in these studies, which could be contrasting to each other.

In the first one, it is believed that contingencies are extracted in a bottom-up way by

a kind of associative learning, classical with ANNs and with simple RNNs. In this case, the standard task is the Serial Reaction Time (SRT) task : The subject is asked to reproduce sequences with hidden regularities and improvements of performance in the reproduction of sequences are measured (Perruchet and Pacton, 2006). It is believed that a perceived symbol is associated to the next symbol to produce, and a reduced reaction time is seen as a clue that the transition to be reproduced is valid. The second approach is top-down and considers that the knowledge is encoded in grammatical rules. The task of Artificial Grammar Learning (AGL) is generally considered in this approach (Pothos, 2007). Grammatical sequences built with a finite state language are presented and it is believed that the RNN is able to extract and build an automaton encoding the corresponding language. This process is less dependent on frequences of local transitions and in the test phase, new strings can be classified as grammatical or not grammatical.

Two contexts for knowledge extraction can appear : When the grammar is known a priori, and when it isn't known or hidden. In the first context, it is possible to generate a learning corpus including positive and negative (valid and non-valid) examples to learn the task as a categorization problem. In addition, the extracted rules or contingencies can be directly compared to the original grammar for validation, considering that valid examples should be recognized and non-valid should be rejected. In the second context (*i.e.* when the grammar is not known or hidden), only positive (valid) examples are available for learning. The task then is predicting the next element (*e.g.* action, decision, symbol) from the current element and the past ones. In this context, assessing the quality of the extracted knowledge is a more difficult process, only possible by experimentation or by subjective analysis by a human expert.

In our work, we will mainly consider the AGL approach. We will keep the grammar hidden in the learning phase, thus corresponding to a prediction task, but we will be able to evaluate directly the results against the grammar in the validation phase. We will also present preliminary work on a real data set, from the domain of electrical diagrams analysis where no grammar is available, thus requiring human evaluation.

1.3 Major steps for knowledge extraction from sequences

Within the AGL approach, the extracted knowledge implicitly encoded in an RNN can take the form of a finite state automaton (FSA). In the deterministic case, it is called Deterministic Finite Automaton (DFA). Automata extraction from RNNs has been described as a three-step process (Wang et al., 2018), following a decompositional method (Remm and Alexandre, 2002), for post-hoc interpretability. Considering that the network is already trained with a learning corpus, the first step consists of, for each item in a test corpus (set of sequences), recording the values of the hidden units in the shape of vectors. We will refer to them as the hidden units' activation patterns or hidden patterns. During the second step, all the collected hidden patterns are analyzed to quantize the hidden state space into a finite set of discrete states that can be seen as important features extracted from the network. The third step is the automaton construction followed by a minimization process of the obtained automaton (Giles et al., 1992). In addition, it is important to perform a validation process by submitting novel valid and non-valid sequences to the automaton, to evaluate its generalization capabilities or by requesting the subjective evaluation of a human expert.

1.3.1 Hidden space quantization

Different approaches were used to quantize the hidden space : 1) considering that each hidden pattern is a state of the desired automaton (Weiss et al., 2017), 2) cutting the hidden space into a limited number of discrete states by a quantization process (Giles et al., 1991, 1992;

Omlin and Giles, 1996), and 3) using clustering techniques like k-means (Zeng et al., 1993), self-organizing maps (Tiño and Sajda, 1995; Blanco et al., 2000) and hierarchical clustering (Cleeremans et al., 1989; Servan-Schreiber et al., 1989; Elman, 1990).

One characteristic shared by these studies is that they mostly use positive and negative examples generated from the Tomita grammars (Tomita, 1982) that induce regular binary languages (sequence of 0 and 1), far from real life choices. Only Tiño and Sajda (1995) and Schellhammer et al. (1998) proposed rule extraction processes from RNN fed with only positive examples composed of non-binary choices.

1.3.2 Automata construction

Many techniques can be used for automata construction once the partitioning is done. They all involve generating the states and outputs (and the output classification if necessary), while input patterns are fed to the RNN. Indeed, by feeding the network with input patterns, it is possible to find the appropriate transitions between the states, and thus the edges of the final automaton (Omlin and Giles, 1996).

In the case of a quantization process, the hidden space is divided into "macrospaces" that represent "basically what the rule extraction algorithm "sees" of the underlying RNN" (Jacobsson, 2005) and transitions between macrostates will correspond to the edges of the final automaton.

Another technique is the sampling-based approach, the principle of which is to replace the search in a quantized state space by the recorded activity of the RNN interacting with its environment and the data (Watrous and Kuhn, 1992). Tiño and Sajda (1995) proposed a similar approach but only for the extraction of a Mealy machine (where the output depends on the input at time t and the current state) as opposed to a Moore machine (where the output depends only on the current state) (Keller, 2001).

Another approach proposed by Schellhammer et al. (1998), whose dataset was composed only of positive examples, involves using the k-means algorithm for clustering the hidden units activation patterns and a process based on frequency analysis for the automaton construction phase. Only transitions whose frequency is above a specific threshold are kept and displayed in the final automaton.

1.3.3 Levels of performance

Even, if all these experiments use different algorithms to analyze the hidden space and build the automaton, they are all similar because they all rely on the same three step process initially defined in (Giles et al., 1992; Omlin and Giles, 1996). A major question is subsequently to wonder about the level of performance of this approach, depending on the RNN analyzed and on the complexity of the underlying grammar.

Wang et al. (2018) have recently proposed an important study about knowledge extraction from RNNs, using this kind of decompositional methods. They consider several kinds of RNNs, namely Elman SRN, Second-order RNN, Multiplicative Integration RNN, LSTM and Gated Recurrent Unit (GRU) RNN. Even if these networks are theoretically equally expressive, recent models have been designed to allow for more efficient learning, with Second-order RNN and Multiplicative Integration RNN introducing weights for second order and multiplicative integration of internal representation and LSTM and GRU introducing gate units that can keep a memory of past events. In both cases, learning can directly apply on more elaborated information, at the price of more weight parameters.

For this reason and to ensure a fair comparison of performances, authors adapt the size of the hidden layers of each model, so that they all have the same number of parameters. These

RNN models are trained on the seven Tomita grammars and, for each model, knowledge is extracted with the classical approach of decompositional methods (vector space of hidden layers partitioned into finite parts, each treated as the states of a DFA subsequently minimized).

The reader is referred to Wang et al. (2018) for the detailed analysis of the results for each Tomita grammar but, in short, authors conclude that "second-order RNN provides the best and most stable performance of DFA extraction on the seven Tomita grammars in general". The authors acknowledge that "it is quite surprising that both LSTM and GRU fail on this DFA extraction task with simple datasets, given the success of these models". They propose that this might be due to "the fact that the information representing the discrete states and their transitions are not completely stored in the recurrent layer" [but also in the gates] and conclude that "extracting DFA from these more complicated models remains an open question".

1.4 Positioning and main contributions

The work reported here is concerned with the extraction of structured knowledge acquired by implicit learning of sequences when the grammar is unknown or hidden and only positive examples are available for learning (Tiño and Sajda, 1995; Schellhammer et al., 1998).
We propose to use RNNs to learn corresponding sequences and exploit decompositional methods as defined in Giles et al. (1992); Omlin and Giles (1996) to extract knowledge from RNNs into FSA, following the three step process. Basic RNN models have mostly been used for early attempts of automata extraction. They often demonstrate some weaknesses in the complexity of sequences they can assimilate. The recent study by Wang et al. (2018) indicates that decompositional methods can be successfully applied to more recent RNNs, being able to cope with more complex grammars, except LSTM.
The main contribution of the present paper is to propose an end-to-end efficient methodology for knowledge extraction from LSTM. In brief, by summarizing the vector space of hidden layers as only their output activity, we solve both problems encountered in Wang et al. (2018). On the one hand, instead of using all the hidden weights, we only select one parameter per hidden unit, thus taming the combinatorial increase of using more parameterized models. On the other hand, the output activity of the hidden units is the result of all internal computations within units, not only including hidden weights but also gates. In the present study, we thus demonstrate experimentally two main points :

- The small-sized hidden representation of LSTM, when associated to classical decompositional methods, carries enough information to allow for the extraction of a fully accurate DFA
- The extracted automaton provides local interpretability about specific predictions, and represents also a substitute of the neural network that can mimic the behavior of the whole network for a specific dataset and thus provides insights into the field of global interpretabilty of RNNs

In the sequel of the paper, we first introduce the grammars that have been chosen for generating the corpus of sequences, derived from the Reber grammar. We then present our methodology by describing the LSTM recurrent network used, the knowledge extraction process and the validation process of the extracted knowledge. Finally, we report experimental results obtained on both artificial grammars and an industrial application in the electrical domain, before discussing the main lessons about this study and its importance in the field of knowledge extraction, and evoking prospective work in the last section. Figure 1 depicts the global experimental approach followed in our work and Table 1 lists the used acronyms.

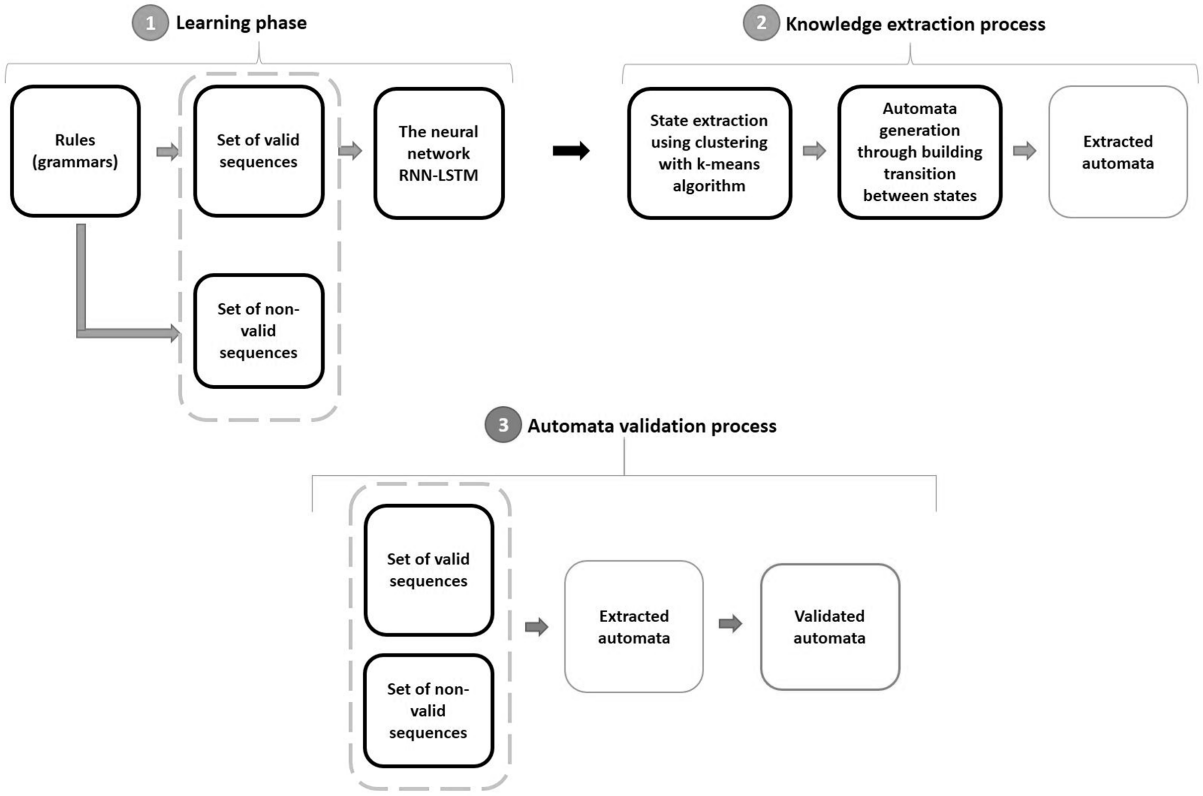

Figure 1. The global experimental approach for implicit knowledge extraction from RNN-LSTM in a task of prediction in three steps : 1) the learning phase where valid sequences generated from a grammar are used to train the network. 2) the knowledge extraction process, and 3) the automata validation process where both valid and non-valid sequences are used. In the text, step 2 and 3 are grouped, but presented visually here separately to explain how sequences are used in the global approach.

Acronym	**Meaning**
ANN	Artificial Neural network
RNN	Recurrent Neural network
LSTM	Long Short Term Memory
RNN-LSTM	a RNN with a hidden layer composed of LSTM units
AGL	Artificial Grammar Learning
RG	Reber Grammar
ERG	Embedded Reber Grammar
CERG	Continious and Embedded Reber grammar
FSA	Finite State Automaton
NFA	Non-deterministic Finite Automaton
DFA	Deterministic Finite Automaton

Table 1. Acronyms used in this paper.

2 Methods

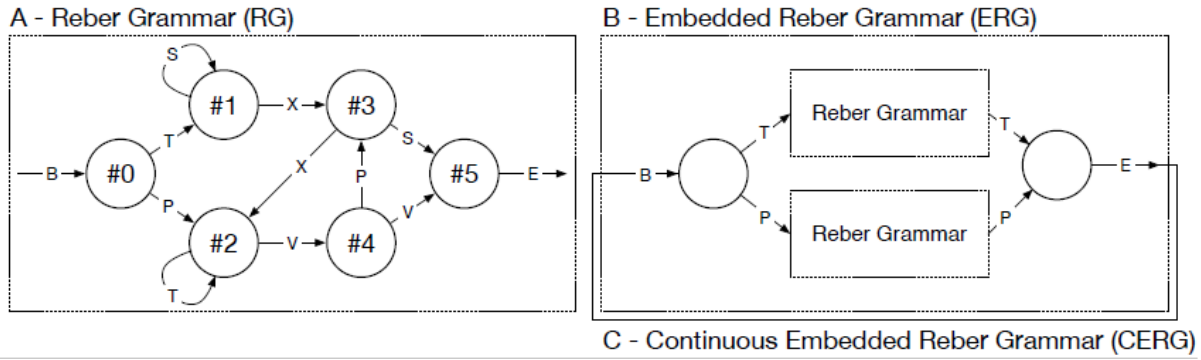

Figure 2. The three grammars used in the experiments, represented as a Finite State Automaton including nodes representing states and bows emitting symbols. From left to right : A - Reber Grammar (RG), B - Embedded Reber Grammar (ERG), C - Continuous Embedded Reber Grammar (CERG). B means "Begin" and E means "End". See section 2.1 for details.

2.1 Reber grammar : Non-binary sequences closer to real life situations

Three grammars were used during this work (cf figure 2): the Reber grammar (RG), a grammar originally used in cognitive psychology experiments about implicit learning ability in humans (Reber, 1967; Cleeremans and McClelland, 1991), as well as two of its variants, the embeded Reber grammar (ERG) and the continuous and embedded Reber grammar (CERG). Even if, as mentionned above, most works in AGL use binary Tomita grammars, we found it more consistent with regard to targeted industrial applications where symbols are electrical components, to use RG, composed of 7 symbols (B,T,P,S,X,V,E) appearing twice, 6 nodes and 12 transitions.

Each variant of RG was employed to test a particular aspect of the RNN, namely performances during prediction, resistance to the size of the sequences and good retention of information when long term dependencies between elements are to be considered. The use of the RG has allowed us to test out the ability of the model to learn and predict in a context where the beginning and end of sequences are easily identifiable. The ERG tests the ability of the network to handle ambiguity for the begin and end symbols that are no longer only at ends but may appear also in the middle of a sequence. Finally, the CERG grammar eliminates the notion of beginning and end. The neural network must learn a continuous flow and be able to make the right predictions based on the local context and the one further back in time. In all of these grammars, transitions between symbols have to be learned in a more or less extended temporal context. Similarly, in a real context such as electrical diagrams analysis, electrical components can be used for different installations according to constraints like amperage, type of power supply and a whole set of electrical technical characteristics, which generate sequences with the same components but with different contexts (Chraibi Kaadoud et al., 2017).

2.2 The procedure of knowledge extraction from LSTM

Among the different variants of the LSTM model (GRU, Vanilla LSTM, LSTM with peephole connexions), we have chosen to implement the version proposed by Gers et al. (2000) (with forget gates and no peephole connection), that the authors applied in their study to the Reber grammar. As presented in figure 3, the network architecture, also inspired by the work of

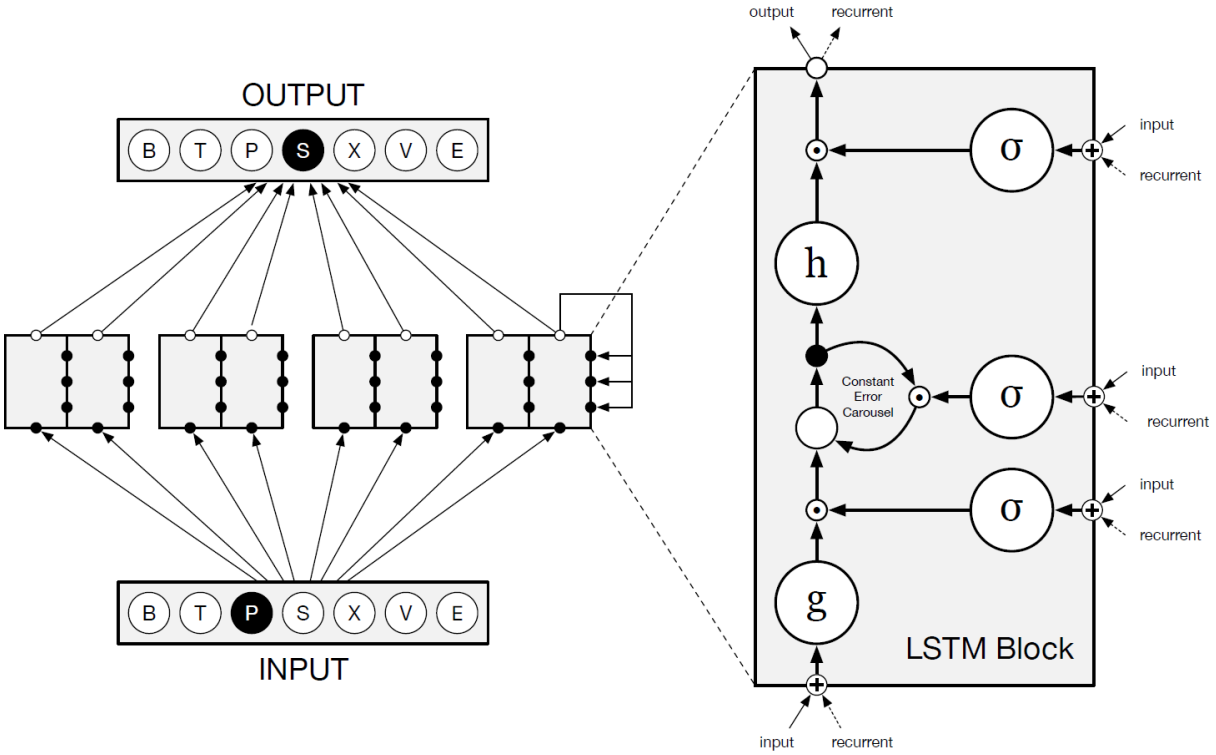

Figure 3. The RNN-LSTM model with three layers. In addition to classical input and output layers, hidden units are introduced : four LSTM blocks with two cells and a CEC (Constant Error Carousel) in each. In the picture, all white dots outside LSTM blocks are linked to all black dots. There are skipped connections between input and output units. The hidden layer provides a real-valued vector of size 8. Figure adapted from Lapalme (2006)

Gers et al. (2000), consists in three layers with recurrences limited to the hidden layer composed of four LSTM blocks with two cells each. For further technical details, B describes the network parameters and characteristics.

The model is trained on the following task : considering a sequence, the network receives as input at time t a symbol in the shape on a binary vector (we use one-hot encoding strategy to encode each symbol) and should predict the symbol at $t + 1$ in the sequence in the shape of a numerical vector (encoding the estimated probability of each symbol). Depending on the sessions, the network is trained and tested not only on sequences but also flows (sequences put end to end) generated by RG, ERG and CERG.

During the learning phase, it is supposed that important knowledge associated to the hidden rules is encoded in the processing part of the network, at the level of the state space of the hidden layer units. This knowledge should allow to define the rules of transition of the grammar (set of rules) and the contexts in which they are valid. Transposing existing approaches for rule extraction from SRNs (Servan-Schreiber et al., 1988) onto LSTMs, at each time step, we record the activity of the outputs of the LSTM cells, considered here as the activity of the hidden layer. We extract implicitly learned features (corresponding to important contexts for rule definition) by a clustering process onto this hidden space and subsequently build an automaton by collecting sequences of the extracted features when valid sequences are fed to the network. The automaton is minimized in the ultimate step, before its evaluation.

2.2.1 Collecting activity patterns of the hidden layer

To extract the knowledge implicitly learned in the hidden space, we generate a test corpus from the same grammar as provided for training the network. To analyze information flow as a sequence is presented, we have created a list of activity patterns, each with a unique identifier ("id") consisting of the current symbol and the time step. In the sequence BTXSE for example, the id list of the activity patterns for each symbol will be : B0, T1, X2, S3. We record the activity of the outputs of the LSTM cells that are represented in numerical vectors of dimension 8.

2.2.2 Feature extraction algorithm using k-means for clustering

We use the k-means algorithm for clustering. This algorithm consists of partitioning the data collected from the hidden space into k groups by minimizing the distance between samples within each partition. The number of clusters in which the data must be partitioned is a parameter of the model. For each value of k, each of the hidden activity patterns ordered in a list that we call "list-patterns" is assigned to 1 among k clusters, thus yielding a list of same size called "list-clusters" indicating the cluster id of each pattern. Consider two examples of lists: A list-patterns of 5 patterns: [h0, h1, h2, h3, h4] and a list-clusters [0 3 2 2 0] from a clustering using k-means with k = 4. An appropriate reading of these two lists is:

- the patterns h0 and h4 belong to cluster 0
- the pattern h1 belongs to cluster 3
- the patterns h2 and h3 belong to cluster 2

Matching both lists, it is possible to associate any pattern to a cluster and thus to a state (*i.e.* a node) in the FSA we wish to extract.

2.2.3 Average Silhouette coefficient analysis for k-means algorithm

To analyze the impact of the clustering parameter, the k value, on the distribution of clusters, we computed the mean silhouette coefficient for all samples for each value of k. The purpose of this method is to study the distance between resulting clusters on a range of [-1, 1](Rousseeuw, 1987). For a given sample, if the silhouette coefficient is close to -1, the sample is assigned to the wrong cluster. If the measure is around 0, the sample is on (or very close to) the decision boundary between two neighboring clusters, and if the measure is close to 1, the sample is assigned to a cluster and far from others. The best k value for clustering is reached for the maximum value of the silhouette coefficient. Computing this value allows thus to objectively evaluate the quality of the clustering computed for each value of k.

2.2.4 Automata generation

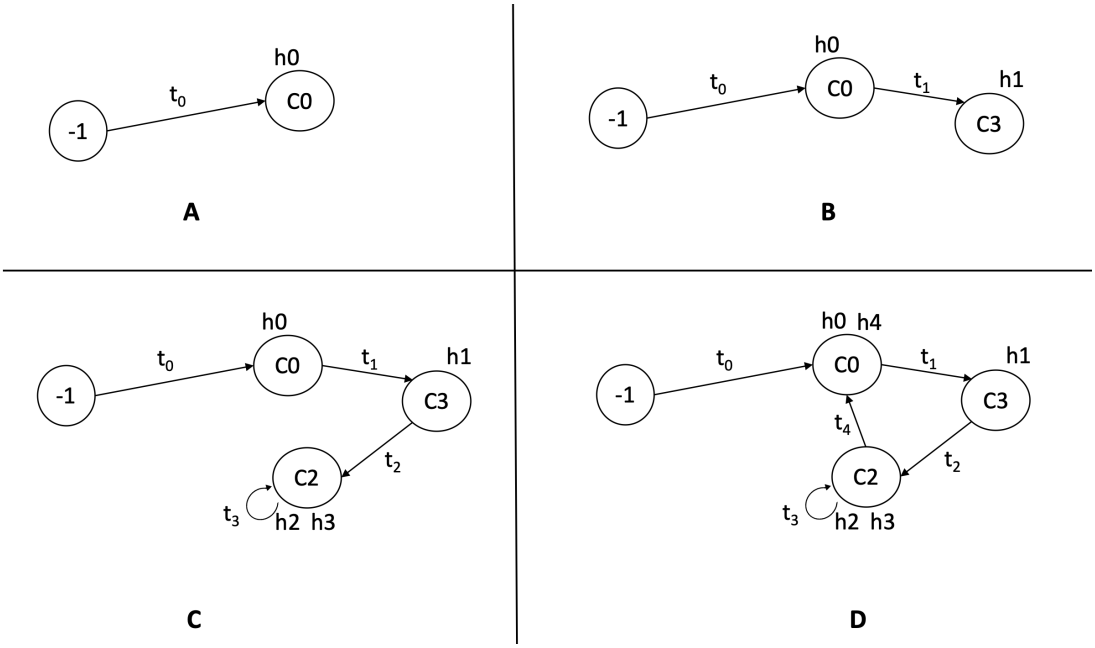

Figure 4. Example of an automaton generation with a list-patterns [h0, h1, h2, h3, h4] and a list-clusters [0 3 2 2 0]. A - Time step t_0, creation of a new node (state). B - Time step t_1, creation a second node (state). C - After time step t_3 and t_4 : h2 and h3 belong to the same cluster, which generates a loop. D - Time step t_4 : h4 belongs to a cluster already visited, which creates an edge between the current node (C2) and the existing node (C0)

We adapted the algorithm described by Omlin and Giles (1996) for automaton generation. Figure 4 presents an example of automaton generation using the five patterns mentioned above and information given by list-patterns and list-clusters. The generation of the FSA is initiated by adding a node with the identifier -1 as a starting point (figure 4, label A).

The rule extraction process requires a simultaneous analysis of both list-patterns and list-clusters : for each pattern, if the associated cluster is a new one (*i.e.* not represented as a node in the FSA), a new node is then added with its id as cluster number, together with a directed edge from the previous node to the new node (figure 4, label A, B, C). If the current pattern belongs to a cluster already represented in the FSA, then a directed edge is added between the previous node and the corresponding node (figure 4, label D, time t_4). In the case where two consecutive patterns belong to the same cluster, a recursive connection is added to the node

representing the cluster (figure 4, label C, time t_3). This process yields an automaton with unlabeled transitions explaining the arrangement of the clusters (states).

To improve the quality of the extracted automaton and give more information on the processing of sequences in the hidden space, we have modified the algorithm proposed in Omlin and Giles (1996), extending the mechanism proposed in Schellhammer et al. (1998) associating nodes with their frequency, to edges and the information they carry. In our approach, each edge is assigned an identifier during the automaton generation phase, corresponding to the symbol that the LSTM processes at the associated time step. In the previous example, if sequence BTXSE is the first to be analyzed by the network, the identifier of each symbol will be B0, T1, X2, S3. If an edge already exists between the two nodes, the new symbol is added to its identifier.

This original process allows to generate an FSA with long labels on edges, that makes the temporal organization of the RNN patterns explicit. Figure 6a shows the example of an automaton generated without a label and figure 6b shows the same automaton with labels. Algorithm 1 describes the process of extracting the rules in the form of an FSA with long labels (figure 6c) using the hidden activity patterns of an RNN-LSTM. Long labels are interesting because they give information about the place of symbols in the sequence. They can also be synthesized into short ones by just keeping the id of the symbol and removing numerical information. The comparison with the original automaton of the corresponding grammar becomes visually easier (for small automata). Figure 6b and Figure 7b present examples of automata generated with a simplified label.

Algorithm 1 Algorithm for extracting rules in the form of a FSA with long labels, using the activity patterns of an RNN-LSTM

```
Require: # Learning and test of the RNN——
  RNN_LSTM.learning(learning_data_set)
  # labels_list : list of symbols presented to the network during tests
  activity_patterns_list, labels_list = RNN_LSTM.test(test_data_set)

  Function rules_extraction (activity_patterns_list, labels_list, k) :
  # Clustering ————————————————————————-
  clusters_list = k_means(k, activity_patterns_list)
  # Generation of automaton A————————————————————————-
  A = {} # Dictionnary
  current_node= -1
  A['nodes'].add(current_node)
  A['edges'] = [ ] # list of dictionnaries
  for all pattern h of index i from activity_patterns_list do
    associated_cluster = clusters_list[i]
    if associated_cluster ∉ A['nodes'] then
      A['nodes'].add(associated_cluster)
    end if
    edge= {} # Dictionnary
    edge['id'] = (current_node, associated_cluster)
    if  edge ∉ GA['edges']  then
      new_edge = edge
      new_edge['weight'] = 1
      new_edge['label'] = labels_list[i]
      A['arcs'].add(nouvel_edge)
    else
      edge['weight'] = edge['weight'] +1
      edge['label'] = edge['label']+ labels_list[i]
      A['edges'].update(edge)
    end if
    # Update of the current node
    current_node= associated_cluster
  end for

  return  A

  End Function
```

2.2.5 Automaton minimization

The last step for automaton generation is the minimization process. Minimization algorithms exist for deterministic finite automaton (DFA). They involve transforming a DFA into a minimal version of that DFA with the minimum number of states. The process starts by determining the type of the FSA : non-deterministic finite automaton (NFA) or DFA. A DFA has transitions that are uniquely determined by the input symbol from each state. On the contrary, an NFA is an automaton where several possibilities of transition can exist simultaneously for a state and a given input symbol. In this case, a conversion into a complete DFA is made.

In our work we use the table-filling method, derived from the Myhill-Nerode theorem (Nerode, 1958), for the minimization process.

This method involves first building a transition table containing as many columns as lines, where each of them represents a state of the automaton. The second step is an iterative process that fills the table according to the transitions of the automaton. The remaining unmarked pairs are grouped as a single state. Algorithm 2 describes this method.

Algorithm 2 The Myhill-Nerode algorithm for DFA minimization

```
Function rules_extraction (final_states, non_final_states):
Draw a table for all pairs of states (P, Q)
Mark all pairs where only one state belongs to the final_states
Repeat until no more marking can be done:
    if there is any unmarked pairs (P,Q) then
        # δ is the transition function that "transform P into P' according an input X
        if [δ(P,X), δ(Q,X)] is marked then
            Mark(P,Q)
        end if
    end if
End Function
```

2.3 Validation procedure and analysis

The last step of the proposed methodology involves analyzing the minimized DFA by feeding it with grammatical sequences in order to confirm if it recognizes the same language as the original grammar. For this purpose, we apply the following process (figure 5) : for each sequence, the starting node is the one containing -1 in its label. We apply the input on it (corresponding to the first symbol of the sequence) to retrieve a new state. We establish then the neighbors list (*i.e.* states) of this new state and their associated transitions. If among these transitions, there is one corresponding to the next symbol of the sequence, the new state becomes the current state. The process is then repeated again, until the next symbol of the sequence is the last symbol of the sequence (*i.e.* symbol E). In this case, we check if among the successors there is the beginning symbol (*i.e.* symbol B, figure 6c). If this condition is true, the minimized DFA not only recognizes the full sequence, but also respects the long term dependencies of the original grammar. If among the transitions of the neighbors, none of them corresponds to the desired next symbol, the sequence is rejected. For each k value of the clustering, we measure the average percentage of accepted sequences for 10 simulations on different grammatical sequences.

We emphasize here the difference between our validation approach and those widely used in the literature. As our learning corpus only involves positive examples, we cannot reproduce the validation process used in Wang et al. (2018), which consists in testing positive and negative examples on DFA. Here, we verify that the rules structuring the Reber grammar are indeed those encoded by the network, *i.e.* that the succession of symbols in a precise order as well as the sequential dependencies are preserved. This implies that the local context of a prediction is well learned and that the global representation of the network behavior over time is adequate with the original grammar. Adopting such an approach thus allows to validate the implicit encoding power of LSTMs.

3 Results

All the simulations were run on a Macbook Pro using the Python scientific stack, namely Numpy (van der Walt et al., 2011), Scipy (Jones et al., 2001), Matplotlib (Hunter, 2007) and scikit-learn (Pedregosa et al., 2011).

3.1 Artificial data set

Three different LSTM-RNNs have been trained on data sets generated from each of the three grammars. The training corpuses are described in Table 2 and A describes the sequence generation algorithm used to build the data sets, for RG, ERG and CERG for reproducibility purpose. We have first checked that we could replicate the level of performance reported in Gers

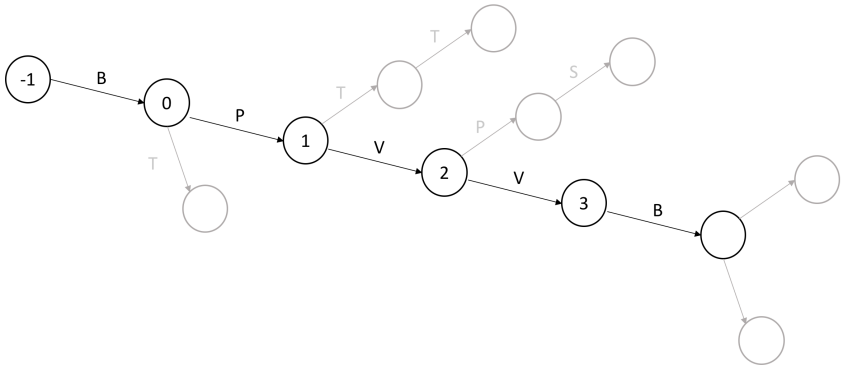

Figure 5. Schematic representation of the testing process of the original sequence BPVVE from the Reber Grammar on the extracted and minimized DFA. In black the selected path on the minimized DFA corresponding to the sequence, in gray the ignored ones.

	RG	ERG
Learning Corpus		
Size	200 000	200 000
Number of samples	1 397 109	2 198 671
Average length	7.98	11.99
Standard deviation	3.35	3.37
Corpus of valid test sequences		
Size	20 000	20 000
Average number of samples	140 193	219 411
Average length	8.02	11.97
Standard deviation	3.47	3.35
Corpus of non-valid test sequences		
Size	130 000	130 000
Average number of samples	140 538	219 533
Average length	8.00	7.01
Standard deviation	6.51	5.50

	CERG
Learning Corpus	
Number of streams	30 000
Number of symbol per stream	100 000
Test corpus (after the learning of each training stream)	
Number of streams	10
Number of symbol per stream	100 000

Table 2. Characteristics of the datasets. On the left, for RG and ERG, numbers concerning the test set are averages calculated on 10 corpuses. On the right, for CERG, size of streams used for training and test phase.

et al. (2000) for the RG and ERG grammars, with a rate of acceptance of valid sequences and rejection of non-valid sequences over 99% (test corpuses also described in Table 2), thus confirming that our LSTM-RNN model successfully learned the grammars. This table also indicates that 10 simulations have been carried out for the test phase and for each simulation, the hidden patterns for 20 000 valid sequences have been recorded, to be used later on for automata extraction, for the RG and ERG grammars (see below). Concerning CERG grammar, we couldn't follow the same procedure for the testing phase with percentage of sequences accepted and rejected, because this grammar proposes a unique continuous flow (and no separated sequences). We consequently replicate the criterion used in Gers et al. (2000), requiring in the testing phase 10^6 consecutive symbols without errors to validate the learning. To carry out this test, we alternate learning and prediction: We make the network learn a flow of 100,000 successive symbols at the end of which we freeze the weights. These weights are used for a test phase involving 10 streams of 100,000 symbols. The average number of prediction errors was measured at each test phase. We succeeded in testing 30,000 training flows without error, thus reaching the Gers criterion. In the following sections, we present FSA extraction results only on RG and ERG.

3.1.1 FSA Extraction : visual results for interpretability

Before evaluating more quantitatively the extracted FSA, we first propose here a more qualitative evaluation, showing their illustrative capabilities. All figures in this section present FSA before minimization, extracted from a small number of hidden patterns. In figures 6a and 6b, we represent a simplified version of the graph, without the loops labelled with the symbol B from the final node to the starting node.

In the RG context, by following the methodology described in section 2.2.4, we obtain, for k=10 clusters, the FSA without label represented in figure 6a, the FSA with long labels represented in figure 6c and the final automaton graph in figure 6b. These results were extracted from the analysis of the first 33 time steps (*i.e.* the first 33 hidden patterns recorded) during the test phase. To help analyze results, note that it was observed a posteriori that the data set was composed of several occurrences of the following sequences : BPVVE, BTXSE, BPTVVE, BTXXTTTTVVE.

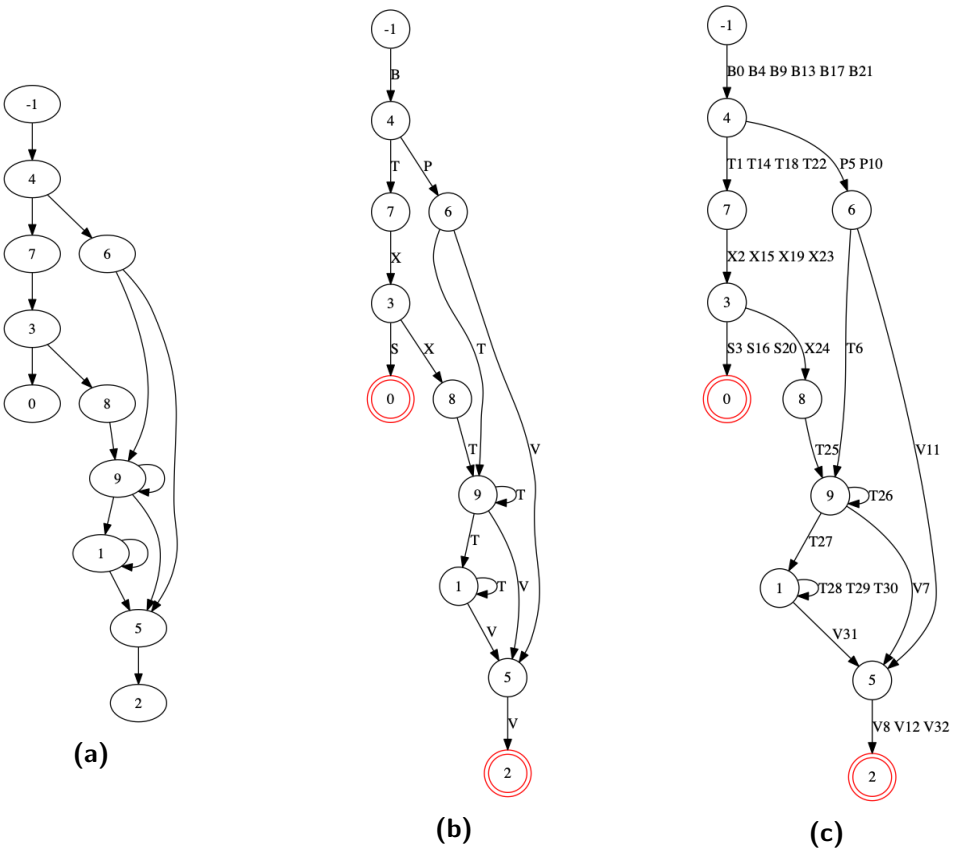

Figure 6. Extraction in RG context on 33 hidden patterns : An unlabeled FSA (a), a final FSA (b) a long-label FSA (c) obtained with a k-means algorithm for clustering, where k=10. Final nodes, that indicate the end of sequences (*i.e.* that the following symbol is E), are noted with red double circles.

In the ERG context, an extraction process was realized on the first 30 hidden patterns as presented in Figures 7. It was also observed during the a posteriori analysis process that those patterns were related to 4 sequences: BPBTSXSEPE, BPBPTTVVEPE, BTBPVVETE, BPBPTVVEPE. We obtain, for k=10 clusters, one FSA with 3 different notation systems: the FSA without label, represented in figure 7a, the FSA with long labels represented in figure 7c and the final FSA in figure 7b.

In the case of testing the model on a small volume of data, the extracted FSA will not

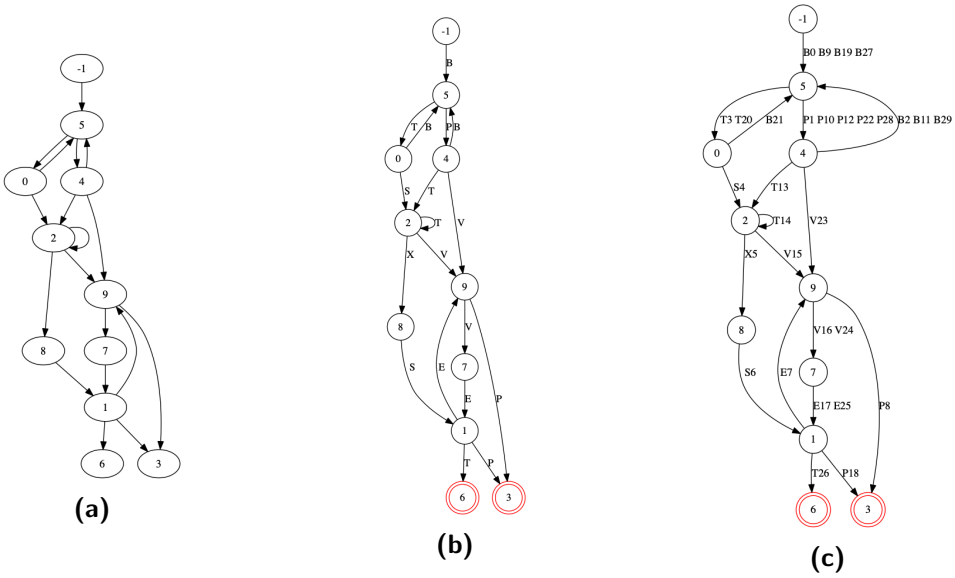

Figure 7. Extraction in ERG context on 30 hidden patterns : An unlabeled FSA (a), a final FSA (b) a long-label FSA (c) obtained with a k-means algorithm for clustering, where k=10. Final nodes are noted with red double circle.

represent all the implicit and encoded representation of all the learned data, but just the part of the representation that corresponds to those inputs. This is why in figures 6, 7, 8, 9 and 12 some of the loops and the transitions that are originally present on the RG and ERG may be absent.

The main result to underline in this section is the visual interpretability that the labels on transition provide. In figures 8 and 9, we present a comparison between the extracted FSA which represents the implicit representation as encoded by the network and the portion of the original grammar that generated the sequences that help build the representation. On top of that, both figures 6c and 7c explicit the temporal behavior of the model. In other words, these knowledge representations allow to explain the reasons of the behavior of the network at a specific time step. It is thus valuable for local interpretability in RNNs.

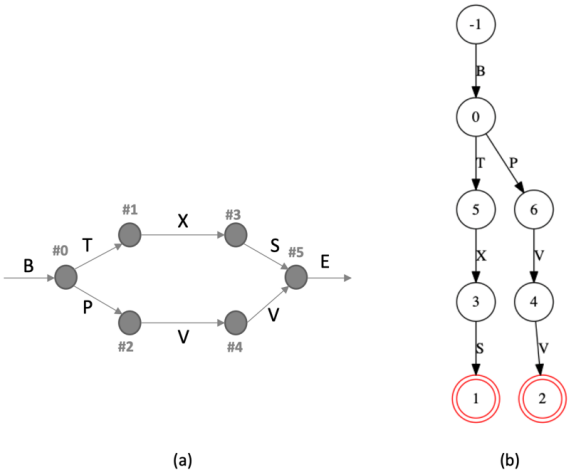

Figure 8. Comparison of a portion of RG (a) and an extracted FSA for k=9 (b) for the 15 first time steps related to occurrences of 2 sequences: BPVVE and BTXSE. Final nodes are noted with red double circle on the extracted FSA.

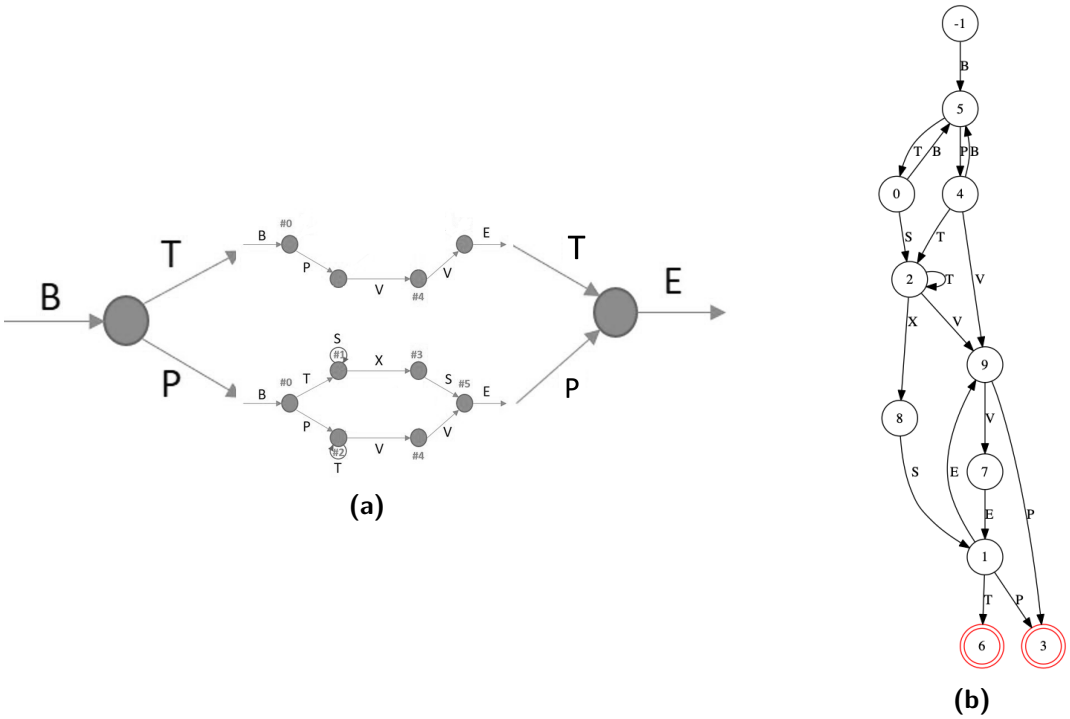

Figure 9. Comparison of a portion of ERG (a) and an extracted FSA for k=10 (b) for the 30 first time steps related to occurrences of 4 sequences : BPBTSXSEPE, BPBPTTVVEPE, BTBPVVETE, BPBPTVVEPE. Final nodes are noted with red double circle on the extracted FSA.

3.1.2 FSA Extraction : evaluation of the extracted FSA using valid sequences

To evaluate the quality of the results, we analyze: 1) the evolution of the average silhouette coefficient for each k value (only the results of analyses of the first 5000 patterns are shown, unless stated otherwise) and 2) the percentage of valid sequences recognized by each minimized DFA for each k value.

Figures 10 and 11 represent, for each value of k, the evolution of the average silhouette coefficient and the percentage of recognized valid sequences in RG and ERG contexts respectively. The maximal value for the average silhouette coefficient is for k close to 400 in both contexts. Nevertheless, the minimized DFA recognizes more than 70% of the valid sequences from k $>$ 50. In other words, when it comes to select the best k value as indicated by the maximal value of the average silhouette coefficient according to figures 10 and 11, k should be close to 400 which will require a lot of computation. But it seems also possible to choose k=50 to observe good results with much less computation.

A related important information discussed below is that in the former case, the computing time counts in days not to say weeks, whereas in the latter case, some minutes are sufficient. If we compare the evolution of the percentage of accepted sequences to the silhouette coefficient analysis of k in both grammars, it appears that the higher the silhouette coefficient is, the more it is possible to get an extracted minimized DFA that can recognize the original sequences and accordingly the original rules (and thus the original grammar) hidden in those sequences.

We repeated the extraction process on 5000 different patterns 10 times, and we obtained the following results (average values) :

For RG context :

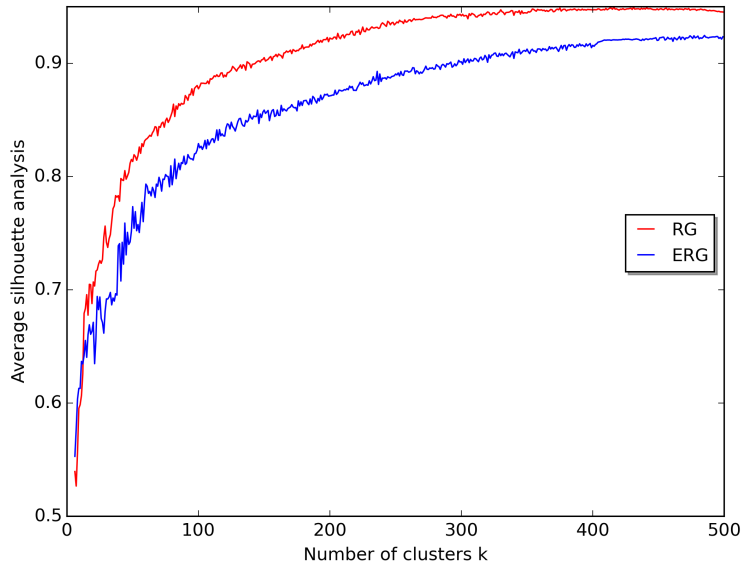

Figure 10. Analysis of extraction process of 5000 patterns for k in [6,500] : Evolution of the average silhouette coefficient in RG context in red and ERG context in blue .

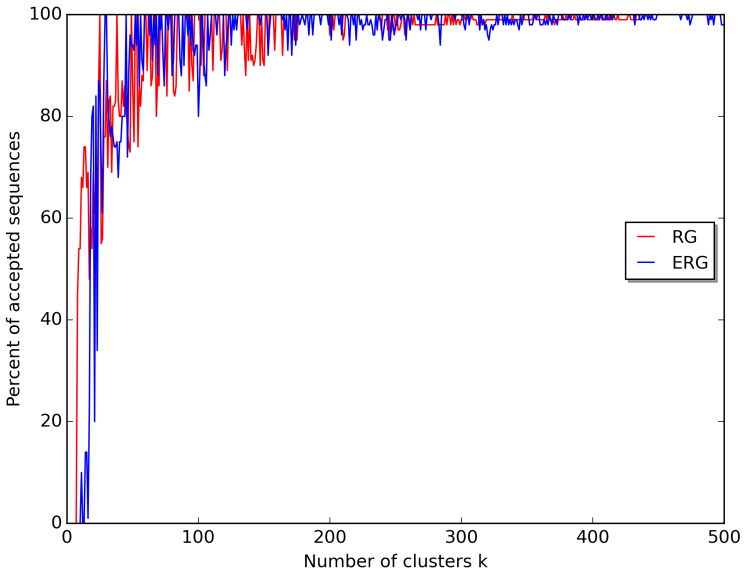

Figure 11. Analysis of extraction process of 5000 patterns for k in [6,500] : Evolution of the percent of accepted sequences in RG context in red and ERG context in blue.

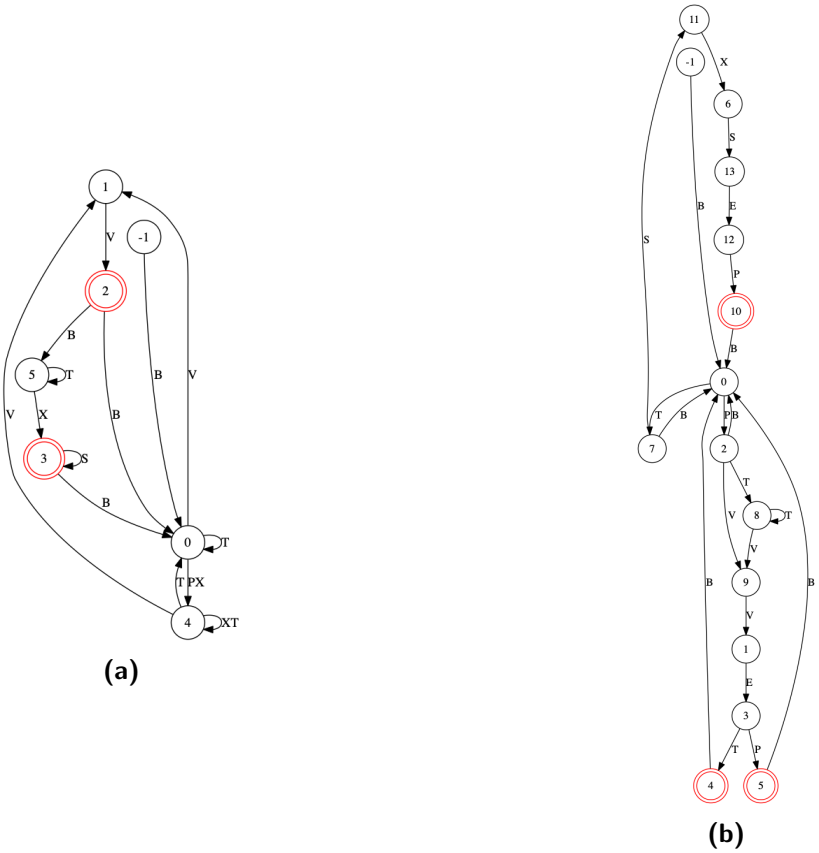

Figure 12. Comparison of extracted FSAs for k=6 for RG (a) and for k=14 for ERG (b). The presented FSAs are not valid for both context, but show here that the selection of the adequate parameter k is independent of the number of nodes present in the original grammars. Choosing k equal to that number does not guarantee that the FSA will be valid.

- for k $>$ 50, average value of silhouette coefficient is $>$ 0.80 and the percentage of accepted sequences is between [70; 100]
- for k $>$ 100, average value of silhouette coefficient is $>$ 0.875 and the percentage of accepted sequences is between [85; 100]

For ERG context :

- for k $>$ 50, average value of silhouette coefficient is $>$ 0.750 and the percentage of accepted sequences is between [85;100]
- for k $>$ 100, average value of silhouette coefficient is $>$ 0.825 and the percentage of accepted sequences is between [87;100]

These results show that it is possible, using the hidden representation of the hidden layer of an RNN-LSTM, to extract a knowledge representation of the hidden rules that fits the original grammar by 80% in the worst case.

On top of that, the analysis of our results shows that our algorithm converges. In the cases of RG and ERG, an increase in k makes it possible to extract automata that recognize a larger number of sequences. Let us underline the importance of this result because it implies that the choice of k is not a limitation of our algorithm, but a compromise to be achieved according to the precision sought after. Indeed, to have more precise results, it is sufficient to increase k (the number of clusters), at the cost of a greater need for computing resources. We also investigated the rather intuitive idea that the k value might equal the number of node in the original grammar for RG and ERG, but both figures 10 and 11 reject this simple idea, since the average silhouette coefficient and the percentage of recognized sequences are at their minimal values in both cases. Figure 12 presents non valid FSA for k=6 for RG and k=14 for ERG.

The results contribute to interpret the k value in the k-means clustering as an adjusting parameter : when k increases, it induces more precise and extended knowledge representation at the cost of lower generalization in the representation. In other words, what is earned in accuracy is lost in generality.

k is therefore a cursor to be adjusted according to the level of interpretability we want for a situation. The methodology we propose here provides flexible results depending on the value of k and the available computing power.

3.2 Real data set (electrical diagrams)

To test the applicability of our approach on real data, we applied our approach on sequences generated from electrical diagrams. Diagrams considered here are large and complex plans of the wiring of industrial installations, that can be represented on documents of hundreds of pages long. Previous works shed light on the existence of hidden knowledge inside sequences of electrical components (Chraibi Kaadoud et al., 2017). For this initial exploration, with the help of experts from the electrical domain, we selected 3 diagrams of similar installations including 25 different electrical components. Following the wiring, we could extract several sequences of such components and, thanks to the equivalence between some components mentioned by the experts, we could automatically generate 200 000 sequences for a learning phase and 20 000 sequences for a test phase.

Following the same procedure as described above, we could train an RNN-LSTM network and could verify an accuracy over 99% in the test phase, thus confirming the good performances of LSTM in sequence learning and prediction. More interestingly, we generated the corresponding FSA and proposed its subjective analysis by the experts. For illustration, the extracted FSA is presented in figure 13, with 25 symbols (electrical components) and 40 states. Although the

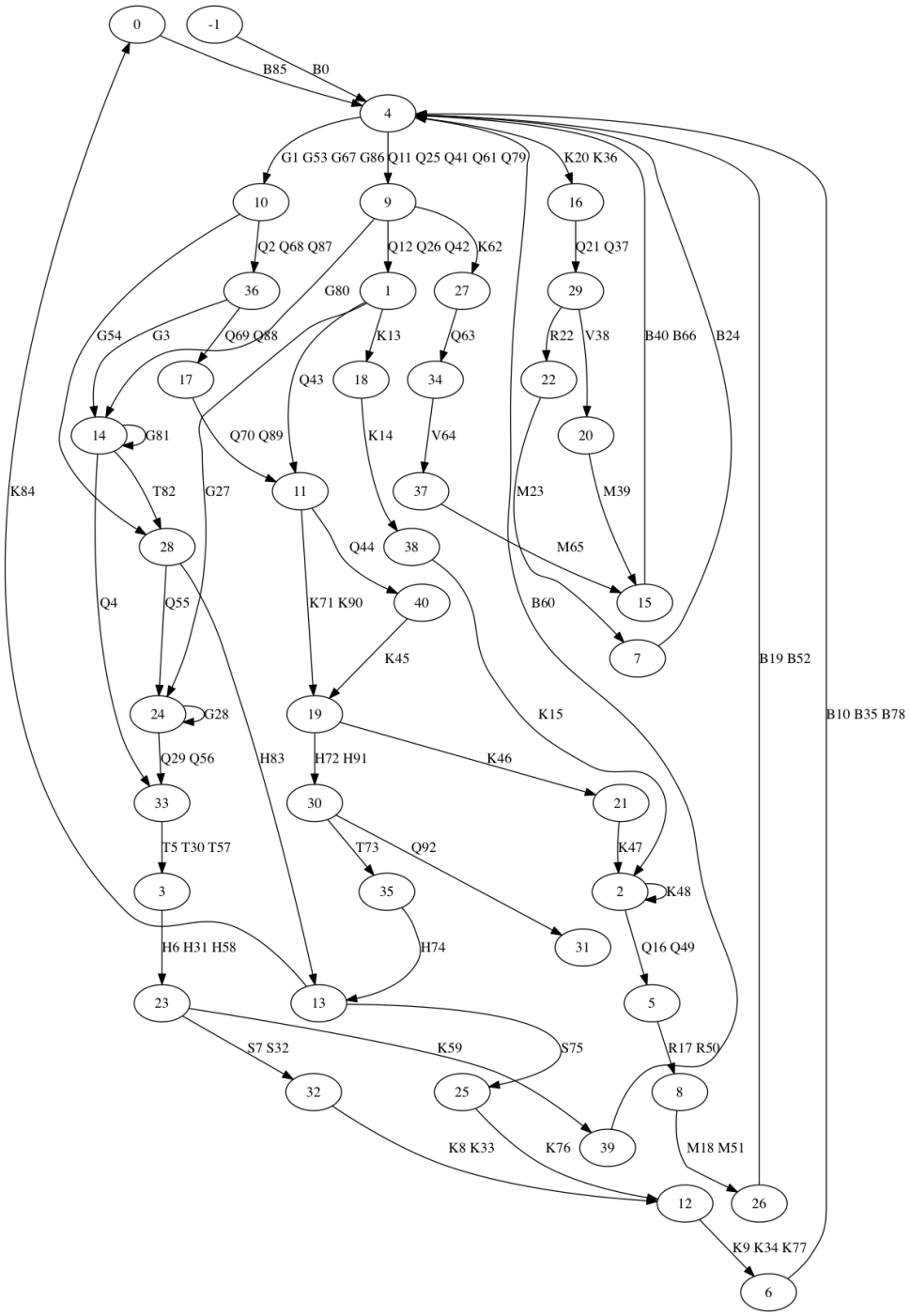

Figure 13. Result of the interpretability approach on electrical sequences from real word electrical diagrams : A long-label FSA obtained with a k-means algorithm for clustering, where k=41 and from an RNN-LSTM having learned sequences of electrical components. Extraction was performed on the first 93 time steps.

evaluation is subjective, it can be reported that the FSA is meaningful for the experts. They found in the automaton a large part of their business knowledge at the level of electrical connectivity rules. An electrical diagram of several hundred pages could therefore be summarized in a single graph on a single page. On our side, these preliminary results showed that our approach transposed on a real context where the grammar is more complex is completely functional and that it especially allows to explain predictions related to an electrical component as a function of the components used before. Nevertheless, it is necessary to put these preliminary experiments into perspective because in practice, the choice of an electrical component is not only due to several characteristics such as its model or its amperage, but also to the habits of the expert and other explicit constraints. We have not had the opportunity to explore whether our approach allows to take into account dependencies of this type, but we believe that this is an interesting line of research to pursue.

4 Discussion

In this paper, we tackle the problem of knowledge extraction acquired by implicit learning of sequences when the grammar is unknown or hidden and only positive examples are available for learning. The main contribution of our work is to propose an end-to-end methodology for knowledge extraction from LSTM. We also underline that this approach could be directly applied to peephole and GRU networks, since it is also possible with these networks to easily get the output activity of the hidden layers, thus extending the interest of our work. At the experimental level, we were able to demonstrate that it is possible to learn complex grammars with LSTM models and to extract knowledge, in the form of a graph that represents rules, acquired by learning from such models, for their interpretability.

Since our work is a multidisciplinary one at the crossroad of several domains, we will discuss our contributions for each domain : implicit learning of temporal behaviour, neural network interpretability and knowledge extraction.

First, let us discuss **the capabilities of implicit learning of temporal behaviour with LSTM**: we could verify that, thanks to their unique characteristics, such as short-term memory (activity patterns at each time step), long-term memory (weights), and intermediate memory (CEC of LSTM units), RNN-LSTM networks are able to learn and extract a stable representation of the encoded rules hidden in sequences, which can be of variable length and include long temporal dependencies between non-binary symbols in three different contexts where the ambiguity of the sequences is increasing.

Then, **at the interpretability level**, it is important to highlight that no precise methodology to extract acquired rules in LSTM models was available despite the fact that these models have demonstrated their learning capabilities. Our work thus allows to fill this gap. To do so, we assumed that by implicit learning, 1) important knowledge about grammatical rules corresponding to the valid transitions and their context in sequences is encoded in the hidden layer of the RNN-LSTM and 2) that it can be extracted and explicitly represented as an automaton. We adapted for the LSTM models the classical three step process already described for simple and more advanced RNNs, by proposing several extensions and improvements, including the use of non-binary grammars and of continuous flows in addition to limited sequences. By only processing the output activity patterns of the hidden layer, we were able to extract a representation in the form of graphs, with different notation systems (cf figures 6, 7), each carrying information on the internal functioning of the RNN-LSTM. The representation without notation informs about the arrangement of states and transitions between them. The notation with long labels informs about the temporal arrangement of patterns between the different states, and offers a contextual explanation regarding the management of patterns by the RNN-LSTM.

Finally, the representation with simplified notation provides a synthetic and explicit representation of the grammatical rules learned, governing the predictions of the RNN-LSTM, to be directly compared with the original grammars. Figure 8 shows a comparison between the Reber grammar and the extracted automaton with simplified notation, and figure 9 the same in an ERG context. The final automaton with simplified labels, once minimized, has been fed with valid sequences for each considered grammar. Over 10 consecutive simulations, the percentage of recognition of valid sequences is above 80% for k $>$ 50 for RG and ERG. In addition, thanks to the use of the silhouette coefficient to monitor the k value, we have shown that this percentage is not a limit in itself of our algorithm, but a compromise to be made between the degree of precision desired during the extraction process and the computing power allocated. The k value in the k-means algorithm is thus an important parameter for the interpretability : rather than accuracy of the extracted automaton, the important point is to determine "what is a good level of representation", which is a context and a human dependent question. Since the use of non-binary grammars was mainly motivated by the exploration of more realistic problems, we have also shown through preliminary results, that our approach could be transposed to more concrete fields, and particularly here the analysis of electrical diagrams, where the grammar is probably more complex and not explicitly known by the experts.

Next, **in the knowledge extraction field**, this work offers a methodology for implicit rule extraction in the shape of automaton when only positive examples are provided. Indeed, this methodology was applied to an industrial context where sequences of electrical components on electrical diagrams that represent industrial installations were available. The first results of our methodology to extract knowledge was submitted to experts and showed that the extracted rules hidden in those sequences did indeed contain not only electrical connectivity rules but also habits of the engineers that draw electrical diagrams (Chraibi Kaadoud et al., 2017; Chraibi Kaadoud, 2018). This makes us propose that, more generally, this process of knowledge extraction could be 1) interesting to study expertise in many fields, which represents a very precious knowledge in many companies but is most of the time implicit and consequently very difficult to transfer to other people, and 2) applied to different domains where sequences are available and the grammar unknown or not completely understood. This latter point is especially important since, we found few works like ours or the ones reported in Tiño and Sajda (1995) and Schellhammer et al. (1998), proposing a process for extracting rules from an RNN fed only with positive sequences composed of non-binary elements. Although this is probably more typical of real-life phenomena, this topic is rarely studied in the recent literature.

To conclude and more globally, all this work addresses the problem of interpretability in ANNs seen as black boxes and especially in RNNs, which are particularly difficult to understand as they process sequences and not just patterns. We have just evoked that our techniques can provide valuable information at different levels of details, for local as well as global interpretability. It is important to stress the fact that the present work allows to extract a representation of the rules expressed by the activity of the network, *i.e.* the portion of the set of acquired implicit rules used to make a prediction. In other words the presented work can be used to zoom in on the network behavior for a given sub-corpus of data and thus compare the representations of several sub-corpuses. Again, the goal here is not to evaluate the behavior of the network, but to extract the implicit rules present in the sequences analysed by the network after the learning phase in the shape of a graph as illustrated in all figures presented in this work. An important perspective is to work on information about the hidden space richer than the activity of the output of the hidden layer. In particular, to explain how LSTMs support sequential dependencies, it would be interesting to explore the hidden representation at the level of activity patterns of LSTM gates, cells, as well as CECs. Note that this question also arises if we apply our approach to variations of standard LSTMs, such as Gated Recurrent Units (GRU) or LSTMs

with peepholes connections. A network with additional hidden layers could also be considered to study the abstraction of the implicit representation encoded by the network from one layer to another. All these characteristics and perspectives may thus participate to improve trust in machine learning algorithms by making them more accessible to as many people as possible.

5 Acknowledgement

This work was initiated as part of a thesis in collaboration with Inria and the Algo'Tech company (Bidart, France), specialized in computer design and drawing software in the field of electrical diagrams. We would like to thank them, especially the company Algo'Tech who financed the thesis, and all the actors who made this work possible.

References

Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198.

Ayache, S., Eyraud, R., and Goudian, N. (2018). Explaining black boxes on sequential data using weighted automata. In Unold, O., Dyrka, W., and Wieczorek, W., editors, *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, volume 93 of *Proceedings of Machine Learning Research*, pages 81–103. PMLR.

Bengio, Y., Courville, A. C., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

Blanco, A., Delgado, M., and Pegalajar, M. (2000). Extracting rules from a (fuzzy/crisp) recurrent neural network using a self-organizing map. *International Journal of Intelligent Systems*, 15(7):595–621.

Chraibi Kaadoud, I. (2018). *Apprentissage de séquences et extraction de règles de réseaux récurrents : application au traçage de schémas techniques. (Sequence learning and rules extraction from recurrent neural networks : application to the drawing of technical diagrams)*. Theses, Université de Bordeaux.

Chraibi Kaadoud, I., Rougier, N., and Alexandre, F. (2017). Implicit knowledge extraction and structuration from electrical diagrams. In Benferhat, S., Tabia, K., and Ali, M., editors, *Advances in Artificial Intelligence: From Theory to Practice - 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part I*, volume 10350 of *Lecture Notes in Computer Science*, pages 235–241. Springer.

Cleeremans, A. and McClelland, J. L. (1991). Learning the structure of event sequences. *Journal of Experimental Psychology: General*, 120(3):235.

Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Comput.*, 1(3):372–381.

Elman, J. L. (1990). Finding structure in time. *Cogn. Sci.*, 14(2):179–211.

Gers, F. A. and Schmidhuber, J. (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Trans. Neural Networks*, 12(6):1333–1340.

Gers, F. A., Schmidhuber, J., and Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Comput.*, 12(10):2451–2471.

Giles, C., Chen, D., Miller, C., Chen, H., Sun, G., and Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 273–281 vol.2.

Giles, C. L., Miller, C. B., Chen, D., Chen, H., Sun, G., and Lee, Y. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Comput.*, 4(3):393–405.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Convolutional networks. In *Deep Learning*, chapter 9, pages 330–372. MIT Press.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE Trans. Neural Networks Learn. Syst.*, 28(10):2222–2232.

Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. (2018). A survey of methods for explaining black box models. *CoRR*, abs/1802.01933.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.*, 9(3):90–95.

Jacobsson, H. (2005). Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Comput.*, 17(6):1223–1263.

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik.

Jones, E., Oliphant, T., and Peterson, P. (2001). SciPy: Open source scientific tools for Python.

Keller, R. M. (2001). Classifiers, acceptors, transducers, and sequencers. *Computer Science: Abstraction to Implementation. Harvey Mudd College*, page 480.

Lapalme, J. (2006). *Composition automatique de musique à l'aide de réseaux de neurones récurrents et de la structure métrique.* PhD thesis, Université de Montréal.

Lipton, Z. (2016). The mythos of model interpretability. In *Proceedings of the ICML Workshop on Human Interpretability in Machine Learning*, pages 1–6.

Nerode, A. (1958). Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544.

Omlin, C. W. and Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52.

Oudeyer, P. (2006). Self-organization in the evolution of speech.

Pascual-Leone, A., Grafman, J., and Hallett, M. (1995). Procedural learning and prefrontal cortex. *Annals of the New York Academy of Sciences*, 769(1):61–70.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perruchet, P. and Pacton, S. (2006). Implicit learning and statistical learning: one phenomenon, two approaches. *Trends in Cognitive Sciences*, 10(5):233–238.

Pothos, E. M. (2007). Theories of artificial grammar learning. *Psychological Bulletin*, 133(2):227–244.

Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of verbal learning and verbal behavior*, 6(6):855–863.

Reber, P. J. (2013). The neural basis of implicit learning and memory: A review of neuropsychological and neuroimaging research. *Neuropsychologia*, 51(10):2026–2042.

Remm, J. and Alexandre, F. (2002). Knowledge extraction using artificial neural networks: application to radar target identification. *Signal Process.*, 82(1):117–120.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Rumelhart, D. and MacClelland, J., editors (1986). *Parallel distributed processing.* MIT Press, Cambridge.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Schellhammer, I., Diederich, J., Towsey, M., and Brugman, C. (1998). Knowledge extraction and recurrent neural networks: An analysis of an elman network trained on a natural language learning task. In *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*, pages 73–78. Association for Computational Linguistics.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1989). Learning sequential structure in simple recurrent networks. In *Advances in neural information processing systems*, pages 643–652.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Mach. Learn.*, 7:161–193.

Servan-Schreiber, D., Cleermans, A., and McClelland, J. (1988). Encoding sequential structure in simple recurrent networks. *School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, CMU-CS-88-183.(November, 1988)*.

Setiono, R. and Liu, H. (1996). Symbolic representation of neural networks. *Computer*, 29(3):71–77.

Tiño, P. and Sajda, J. (1995). Learning and extracting initial mealy automata with a modular neural network model. *Neural Comput.*, 7(4):822–844.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Wang, Q., Zhang, K., Ororbia, I., Alexander, G., Xing, X., Liu, X., and Giles, C. L. (2018). A comparison of rule extraction for different recurrent neural network models and grammatical complexity. *arXiv preprint arXiv:1801.05420*.

Watrous, R. L. and Kuhn, G. M. (1992). Induction of finite-state automata using second-order recurrent networks. In *Advances in neural information processing systems*, pages 309–317.

Weiss, G., Goldberg, Y., and Yahav, E. (2017). Extracting automata from recurrent neural networks using queries and counterexamples. *CoRR*, abs/1711.09576.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Williams, R. J. and Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection science*, 1(1):87–111.

Zeng, Z., Goodman, R. M., and Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Comput.*, 5(6):976–990.

A Sequence generation algorithm

We present in this section the sequence generation algorithm used to build the datasets.

Reber Grammar (RG)

The sequences are generated as follows: each sequence (empty at the beginning) is initialized with a B, and node #0 is designated as the initial node. The directed arcs (arcs with a direction) having a probability of 0.5 to be chosen, the next node is selected randomly among the successors of the current node. Once the node is selected, the label of the chosen arc is added to the sequence. At this point, the result is either BT or BP. Once the next node is selected, the random selection process is repeated among the successors of the current node, and so on until node #5 is reached. When this happens, the sequence is considered finished since node #5 has no arcs to other nodes. The symbol E is then added to the sequence, thus concluding it. Figure 14 reminds the RG structure and Table 3 provides an example of sequences generated from that grammar.

Embedded Reber Grammar (ERG)
The algorithm used for ERG sequence generation is the same as for RG. However it is important to highlight that if the sequence starts with "BT" or "BP" then it automatically ends with "TE" and "PE" respectively. Figure 15.B reminds the ERG structure and Table 16 provides an example of sequences generated from that grammar.
Continuous and embedded Reber Grammar (CERG)
CERG allows to generate a continuous flow of symbols. The sequences are generated as in RG and ERG at the beginning, *i.e.* each sequence (empty at the beginning) is initialized with a B, and node #0 is designated as the initial node. However, in this case, the algorithm does not stop at node #5. It continues until the sequence reaches a size of 100 000 symbols. The dataset for CERG is thus composed of sequences of 100 000 symbols. Figure 15.C reminds the CERG structure and Table 4 provides an example of sequences generated from that grammar.

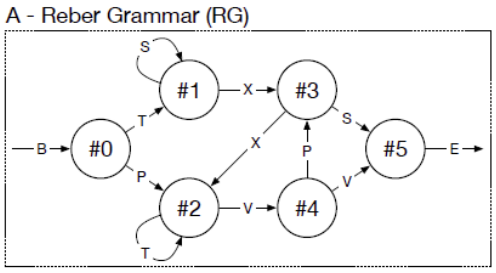

Figure 14. RG

Sequences
BTXSE
BPVVE
BTSXXVPSE
BTSSSSSSSSSSXSE

Table 3. Examples of RG sequences

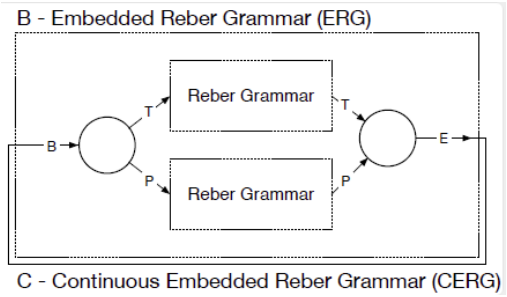

Figure 15. ERG (B) and CERG (C)

Sequences
BTBTXSE**TE**
BPBTXSE**PE**
BTBPVVE**TE**
BPBPVVE**PE**

Figure 16. Examples of ERG sequences

Sequences
BPBTXSE**PEBT**BTXSE**TE BT**BPVVE**TE BP**BPVVE**PE**...
BPBTSSSSSSSSXSE**PEBT**BPVVE**TEBT**BPTTTTTTVPSE**TE**...

Table 4. Examples of portions of CERG sequences

B RNN-LSTM network and learning algorithms

In our work, we implemented a network of three layers and initialized it as described in Gers et al. (2000). In this appendix, we provide first, the technical details related to the network architecture and parameters, and second the algorithms used.

B.1 RNN-LSTM architecture and parameters

The RNN-LSTM consists of 3 layers: an input layer, a hidden layer and an output layer. The input and output layers are both composed of 7 units, due to the fact that the grammars

used have 7 symbols. The hidden layer consists of 4 LSTM blocks each with 2 LSTM cells and 3 gates, *i.e.* 8 cells and 12 gates in total. Figure 3 presents the RNN-LSTM architecture. All the units in the input layer are connected to all the units in the output layer, to each of the 8 cells, and to each of the gates in the 4 blocks. All the cells have connections to each of the output units, but also to the input of all the cells (a cell thus has a recurrent connection to itself) and finally, to each of the gates of each block. For learning, we use a combination of BackPropagation Through Time (Werbos, 1990) and Real Time Recurrent Learning (Williams and Zipser, 1989), that will be defined and explained in the following section. Our network has 424 weights in total. The bias weights of the input and output gates are initialized to -0.5 for the first block, then decremented by -0.5 for the next ones (-1, -1.5, -2). For the forget gates in the 4 blocks, we implemented the reverse : for the first block, the bias of the gate is initialized to 0.5, then is incremented by 0.5 for the next gates in the other blocks (1, 1.5, 2). All other weights, as well as the output layer bias, are initialized randomly between [-0.2, 0.2]. For this implementation, the learning rate is initialized to 0.5 and then multiplied by 0.99 every 100 time steps. During learning phase and prediction, at the beginning of each sequence, the previous states s_{t-1}, as well as the input at the previous time steps are reset to zero. The context is thus reinitialized at the beginning of each sequence.

B.2 RNN-LSTM learning algorithms

As originally proposed by Gers et al. (2000), BackPropagation Through Time (BPTT) (Werbos, 1990) and Real Time Recurrent Learning (RTRL)(Williams and Zipser, 1989) are two algorithms used for the learning phase.

The central idea of **BPTT** is the unfolding of the recurrent neural network in time, so that at each instant it can be assimilated to a feed-forward neural network as the perceptron (Rumelhart et al., 1985). Conceptually, at each time step, the network is copied into a new instance, so there are as many instances of the network as time steps processed in a sequence. For each instance, the network receives an input and the internal state of the previous time step, then provides an output. The errors are computed and the weights are updated using standard backpropagation, with one restriction however: the equivalent weights in all instances must be the same. Spatially, each time step of the unrolled RNN can be seen as an additional layer since the network depends on the order of the presented data and the internal state of the previous time step is taken as input at each time step.

RTRL is a learning algorithm that calculates the exact error gradient at each time step. In other words, this algorithm computes the derivatives of states and outputs with respect to all weights during forward propagation at each time step. It is therefore suitable for online learning tasks. Mathematically simpler than BPTT, it is nevertheless very slow: the more units the network has, the more weights it has, and the longer and more complex the computations are. This drawback makes that the algorithm can only be used on very small networks. A detailed version of RTRL is provided by Giles et al. (1991). For a detailed description of the different learning algorithms of RNN, the reader can refer to Jaeger (2002).

In our work we used BPTT and RTRL, and their truncated versions. A truncation implies that the outgoing errors from a cell or gate will be cut off, even though they serve to modify the incoming weights. Thus, the CEC of a cell becomes the only part of the system through which the error can be backpropagated and maintained forever. This makes LSTM updates efficient without significantly affecting the learning power. To do this: (i)the output units use BPTT , (ii)the output gates use a truncated version of BPTT, (iii)the weights going to the cells, the input gates and the forget gates of the blocks are updated via a truncated version of RTRL

We include below the learning algorithm of the RNN-LSTM inspired from Gers and

Schmidhuber (2001).

Notations
For LSTM blocks and cells :

- j : block of a given hidden layer
- v : cell in a given block
- c_j^v : cell v of block j
- in_j : the input gate of the block j
- ϕ_j : the forget gate of block j
- out_j : the output gate of block j
- $s_{c_j^v}(t))$: state s of the cell v of block j

For the output layer of the network :

- k : an output layer neuron of the network
- e_k : error calculated at unit k of the output layer of the network
- t^k : expected output (value) at unit k

Other notations :

- m : neuron with a link to an LSTM unit
- net_x : activation received by the computational unit x (be it a gate or a cell or an artificial neuron)
- y^m : activity of the calculation unit m
- $w_{(l,m)}$: weight from neuron m to the computational unit l

Algorithm 3 Learning algorithm of a RNN-LSTM

Require: Lets consider the following activation functions :
$f(x) = \frac{1}{1+e^{-x}}$ # sigmoid function between [0,1]
$h(x) = 2 * f(x) - 1$ # sigmoid function between [-1,1]
$g(x) = 4 * f(x) - 2$ # sigmoid function between [-2,2]
and their respective derivatives f', h' and g'
for all size sequence n **do**
  for $i = 0$ à $(n-1)$ **do**
    # Step 1 : forward propagation
    network_entry = sequence[i]
    expected_output = sequence[$i+1$]
    obtained_output = RNN_LSTM.forward_propagation(network_entry)
    error_RNN_LSTM = expected_output - obtained_output
    # Step 2 : back propagation
    RNN_LSTM.back_propagation(error_RNN_LSTM)
  end for
end for

Algorithm 4 Learning a RNN-LSTM: forward propagation

Function forward_propagation (network_entry)
# Forward propagation from the input layer to the hidden layer
for all bloc j **do**
  for all cell c_j^v **do**
    #Calculation of the received activations

$$net_l(t) = \sum_m (w_{(l,m)} \cdot y^m(t)) + \sum_p (w_{(l,p)} \cdot y^p(t-1)) \tag{1}$$

with $l \in \{c_j^v, in_j, \phi_j, out_j\}$
Calculation of activations according to the activation functions

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t)) \tag{2}$$
$$y^{\phi_j}(t) = f_{\phi_j}(net_{\phi_j}(t)) \tag{3}$$
$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t)) \tag{4}$$
$$s_{c_j^v}(t) = g(net_{c_j^v}(t)) \cdot y^{in_j}(t) + s_{c_j^v}(t-1) \cdot y^{\phi_j}(t) \tag{5}$$
$$y^{c_j^v}(t) = h(s_{c_j^v}(t)) \cdot y^{out_j}(t) \tag{6}$$

Calculation of partial derivatives dS^{jv} for cell c, input gate in and forget gate ϕ

$$dS_{c,m}^{jv}(t) = dS_{cm}^{jv}(t-1) * y^{\phi_j}(t) + g'(net_{c_j^v})(t) * y^{in_j}(t) * y^m(t-1) \tag{7}$$
$$dS_{in,m}^{jv}(t) = dS_{in,m}^{jv}(t-1) * y^{\phi_j}(t) + g(net_{c_j^v}(t)) * f'_{in_j}(net_{in_j}(t)) * y^m(t-1) \tag{8}$$
$$dS_{\phi,m}^{jv}(t) = dS_{\phi,m}^{jv}(t-1) * y^{\phi_j}(t) + h(s_{c_j^v}(t)) * f'_{\phi_j}(net_{\phi_j}(t)) * y^m(t-1) \tag{9}$$

  end for
end for
# forward propagation from the hidden layer to the output layer
error_RNN_LSTM = vector of size k
for all unit k of the output layer **do**
  # Calculation of the network output

$$net_k(t) = \sum_m (w_{(k,m)} \cdot y^m(t)) + \sum_p (w_{(k,p)} \cdot y^p(t-1)) \tag{10}$$
$$y^k(t) = f_k(net_k(t)) \tag{11}$$
$$error\_RNN\_LSTM[k] = y^k(t) \tag{12}$$

end for
return error_RNN_LSTM
End function

Algorithm 5 Learning algorithm of a RNN-LSTM: back propagation

Function back_propagation (error_RNN_LSTM) :

Back propagation

for all unit k of the output layer **do**

$$e^k(t) = error\_RNN\_LSTM[k] \tag{13}$$

$$\delta_k(t) = f'_k(net_k(t)) * (e^k(t)) \tag{14}$$

end for

for all bloc j **do**

for all cell c_j^v **do**

Between the k units of the output layer and the output gates of the j blocks

$$\delta_{out_j}(t) = f'_{out_j}(net_{out_j}(t)) * (\sum_{v=1}^{S_j} h(s_{c_j^v}(t)) * (\sum_m w_{kc_j^v} * \delta_k(t))) \tag{15}$$

Between the units k of the output layer and the gates **inputs**, the gates **forgets**, as well as the **cells** of the blocks j

$$e_{c_j^v}(t) = y^{out_j} * h'(s_{c_j^v}(t)) * (\sum_m w_{kc_j^v} * \delta_k(t))) \tag{16}$$

end for

end for

Updating the weights of a RNN-LSTM network

$$\Delta_{lm}(t) = \alpha * (\delta_l(t)) * y^m(t), l \in \{k, i, out\} \tag{17}$$

$$\Delta_{c_j^v,m}(t) = \alpha * e_{c_j^v}(t) * dS^j v_{c,m}(t) \tag{18}$$

$$\Delta_{in,m}(t) = \alpha * (\sum_{v=1}^{S_j} e_{c_j^v}(t) * dS^j v_{in,m}(t)) \tag{19}$$

$$\Delta_{\phi,m}(t) = \alpha * (\sum_{v=1}^{S_j} e_{c_j^v}(t) * dS^j v_{\phi,m}(t)) \tag{20}$$

$$W_{zm}(t) = W_{zm}(t) + \Delta_{zm}(t), z \in \{k, i, out, c_j^v, in, \phi\} \tag{21}$$

Fin Fonction