Neural Computing and Applications
https://doi.org/10.1007/s00521-018-3817-2

ORIGINAL ARTICLE

A tree-BLSTM-based recognition system for online handwritten mathematical expressions

Ting Zhang[1] · Harold Mouchère[2] · Christian Viard-Gaudin[2]

Received: 16 July 2018 / Accepted: 6 October 2018

Abstract
Long short-term memory networks (LSTM) achieve great success in temporal dependency modeling for chain-structured data, such as texts and speeches. An extension toward more complex data structures as encountered in 2D graphic languages is proposed in this work. Specifically, we address the problem of handwritten mathematical expression recognition, using a tree-based BLSTM architecture allowing the direct labeling of nodes (symbol) and edges (relationship) from a graph modeling the input strokes. One major difference with the traditional approaches is that there is no explicit segmentation, recognition and layout extraction steps but a unique trainable system that produces directly a stroke label graph describing a mathematical expression. The proposed system, considering no grammar, achieves competitive results in online math expression recognition domain.

Keywords Mathematical expression recognition · Tree-based BLSTM · Local CTC · Online handwriting

1 Introduction

A visual language is defined as any form of communication that relies on two- or three-dimensional graphics rather than simply (relatively) linear text [1]. Mathematical expressions (Fig. 1), plans and musical notations are commonly used cases of visual languages [2]. As an intuitive and easily (relatively) comprehensible knowledge representation model, mathematical expressions could help the dissemination of knowledge in some related domains and therefore are essential in scientific documents. Currently, common ways to input mathematical expressions into electronic devices include typesetting systems such as LaTeX and mathematical editors such as the one embedded in *MS Word*. But these ways require that users could hold a large number of codes and syntactic rules or handle troublesome manipulations with keyboards and mouses as interface. As another option, being able to input mathematical expressions by hand with a pen tablet, as we write them on paper, is a more efficient and direct mean to help the writing of scientific documents.

Handwritten mathematical expression recognition is an appealing topic in pattern recognition field since it exhibits a big research challenge and underpins many practical applications. Difficulties are related to the large set of symbols (more than 100) with many overlapping classes and because of the 2-dimensional (2-D) structures between pairs of symbols (superscript, subscript, fraction etc.). With regard to the application, it offers an easy and direct way to input MEs into computers and therefore improves productivity for scientific writers. We usually divide handwritten MEs into online and offline domains. In the offline domain, data is available as an image, while in the online domain it is a sequence of strokes, which are themselves sequences of points recorded along the pen trajectory. Compared to the offline ME, time information is available in online form. This paper will be focused on online handwritten ME recognition.

Generally, three tasks are involved in ME recognition [3, 4]: (1) symbol segmentation, which consists in grouping strokes that belong to the same symbol; (2) symbol recognition, the task of labeling the symbols to assign each of them a symbol class; (3) structural analysis, its goal is to

✉ Ting Zhang
ting.zhang@mail.ccnu.edu.cn

[1] National Engineering Research Center for E-learning, Central China Normal University, Wuhan, China

[2] LS2N/IPI - UMR CNRS 6004, Université de Nantes, Nantes, France

Published online: 19 October 2018

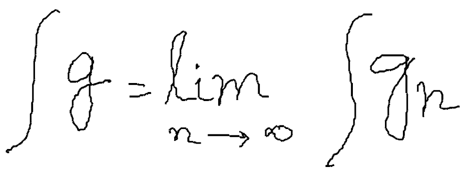


Fig. 1 Handwritten mathematical expression example

identify spatial relations between symbols and with the help of a grammar to produce a mathematical interpretation. The state-of-the-art solutions are mainly grammar-driven solutions: a set of symbol hypotheses may be generated and a structural analysis algorithm (grammar parsing usually) may select the best hypotheses while building the structure. However, these classical grammar-driven solutions require not only a large amount of manual work for defining grammars, but also a high computational complexity for the grammar parsing process. Using a grammar is not mandatory; it can help finding a correct solution when working in a controlled environment. But the problem of fixing a grammar is context dependent and less general than enabling the system to generate independently its own language model. As an alternative approach, we propose to explore a non-grammar-driven solution for recognizing math expressions. This is the main goal of this work; we would like to propose new architectures for mathematical expression recognition with the idea of taking advantage of the recent advances in recurrent neural networks.

Advanced recurrent neural network—long short-term memory (LSTM)—achieved great success in temporal dependency modeling for chain-structured data, such as texts and speeches [5–7]. This success is due to LSTM's representational power and effectiveness at capturing long-term dependency in a sequence. Recently, research on LSTM has been pushed beyond sequential structures. It was extended to tree structures in [8, 9] and DAG (directed acyclic graph) structures in [10]. In this work, we put efforts on a similar task, generalizing the classical LSTM architecture to a tree network topology named tree-based BLSTM. Thereby, the new topology could handle tree-structured data as well as sequence-structured data.

In this work, we have been exploring online handwritten ME recognition from a new perspective, treating it as a problem of deriving a graph from the raw input, and then, extracting several trees which are embedded in the graph and recognized them with a tree-based BLSTM. As known, it is possible to describe a ME using a Stroke Label Graph (SLG, refer to Sect. 2.2.2) in which nodes represent strokes whereas labels on the edges encode either segmentation or layout information. In [11, 12], the solution of building a graph by merging multiple 1D sequences of labels produced by a sequence labeller was proposed and explored. We extend these preliminary works by integrating multiple trees labeled by a tree labeller to recognize MEs.

In Sect. 2, the related works will be reviewed, including the state of the art of ME recognition and representation, and an overview of LSTM. Tree-based BLSTM, as the base of our recognition system, is described in detail in Sect. 3. Afterward, we introduce our tree-based BLSTM recognition system in Sect. 4 step by step, which is the main part of our study. At last, experiments and conclusion are covered in Sects. 5 and 6, respectively.

2 Related works

2.1 Mathematical expression recognition

Research on the recognition of math notation began in the 1960s [13], and several research publications are available in the following thirty years [14–16]. Since the 1990s, with the large developments of touch screen devices, this field has started to be active, producing amounts of research outcomes and gaining considerable attention from the research community. A number of surveys [3, 4, 17–19] summarize the proposed techniques for math notation recognition. This research domain has been boosted by the competition on recognition of handwritten mathematical expressions (CROHME) [4], which began as part of the International Conference on Document Analysis and Recognition (ICDAR) in 2011. It provides a platform for researchers to test their methods and compare them and then facilitate the progress in this field. In this paper, the provided data and evaluation tools from CROHME will be used and results will be compared to participants.

As described already in Sect. 1, ME recognition involves three interdependent tasks [3]: (1) symbol segmentation; (2) symbol recognition; (3) structural analysis. These three tasks can be solved sequentially or jointly. The proposed solutions can be roughly divided into *sequential solutions* and *integrated solutions*. In addition, with recent advances in deep learning, several *end-to-end deep learning-based systems* were proposed for ME recognition. In the coming paragraphs, we introduce shortly these three types of proposed solutions for ME recognition.

Sequential solutions In the early stages of the study, most of the proposed solutions [20–30] are sequential ones which treat the recognition problem as a two-step pipeline process; first symbol segmentation and classification, and then structural analysis. The task of structural analysis is performed on the basis of the symbol segmentation and classification result. Considerable works are done dedicated to each step. For segmentation, the proposed methods include minimum spanning tree (MST)-based method [24], Bayesian framework [30], graph-based method [23, 27] and so on. The symbol classifiers used consist of nearest neighbor, hidden Markov model, multilayer perceptron, support vector machine, recurrent neural networks and so on. For spatial relationship classification, the proposed features include symbol bounding box [13], relative size and position [31], and so on. The main drawback of these sequential methods is that the errors from symbol segmentation and classification will be propagated to structural analysis. In other words, symbol recognition and structural analysis are assumed as independent tasks in the sequential solutions. However, this assumption conflicts with the real case in which these three tasks are highly interdependent by nature. For instance, human beings recognize symbols with the help of structure, and vice versa.

Integrated solutions Considering the natural mutual relationship between the three tasks, researchers mainly focus on integrated solutions recently, which performs the task of segmentation at the same time build the expression structure: a set of symbol hypotheses maybe generated and a structural analysis algorithm may select the best hypotheses while building the structure.

The integrated solutions use contextual information (syntactic knowledge) to guide segmentation or recognition, preventing from producing invalid expressions like $[a+b)$. These approaches take into account contextual information generally with grammar (string grammar [32–36] and graph grammar [37, 38]) parsing techniques, producing expressions conforming to the rules of a manually defined grammar. Either string or graph grammar parsing, both have a high time complexity. Instead of using grammar parsing technique, the new architectures proposed in this paper include contextual information with bidirectional long short-term memory which can access the content from both the future and the past in an unlimited range.

End-to-end neural network based solutions Inspired by recent advances in image caption generation, some end-to-end deep learning-based systems were proposed for ME recognition [39, 40]. These systems were developed from the attention-based encoder–decoder model which is now widely used for machine translation. They decompile an image directly into presentational markup language such as LaTeX. However, considering we are given trace information in the online case, despite the final LaTeX string, it is interesting to decide a label for each stroke. This information is not available now in end-to-end systems.

2.2 Mathematical expression representation

Structures can be depicted at three different levels: symbolic, object and primitive [41]. In the case of handwritten ME, the corresponding levels are expression, symbol and stroke.

In this section, we will first focus on symbol relation tree (SRT), one of the representation models for math expression at the symbol level. From the SRT, when going down to the stroke level, a stroke label graph (SLG) could be derived, which is the current official format to represent the ground truth of handwritten math expressions and also for the recognition outputs in Competitions CROHME.

2.2.1 Symbol level: symbol relation (layout) tree

When only a syntactic representation of a ME is required, operator trees based on symbol operators are adequate tools to describe the ME, but if layout is concerned, symbol relation (or layout) trees are more appropriate. Symbol layout tree represents the placement of symbols on baselines (writing lines) and the spatial arrangement of the baselines [3]. In SRT, nodes represent symbols, while labels on the edges indicate the relationships between symbols. For example, in Fig. 2a, the first symbol '$-$' on the base line is the root of the tree; the symbol 'a' is *Above* '$-$' and the symbol 'c' is *Below* '$-$'. In Fig. 2b, the symbol 'a' is the root; the symbol '$+$' is on the *Right* of 'a'.

101 classes of symbols are present in CROHME data set, including digits, alphabets, operators and so on. Six spatial relationships are defined in the CROHME competition; they are: *Right*, *Above*, *Below*, *Inside* (for square root), *Superscript*, *Subscript*.

2.2.2 Stroke level: stroke label graph

SRT represents math expression at the symbol level. If we go down at the stroke level, a stroke label graph (SLG) can be derived from the SRT. In SLG, nodes represent strokes, while labels on the edges encode either segmentation information or symbol relationships. Relationships are defined at the level of symbols, implying that all strokes (nodes) belonging to one symbol have the same input and output edges. Consider the simple expression $2+2$ written using four strokes (two strokes for '$+$') in Fig. 3a. The corresponding SRT and SLG are shown in Fig. 3b and Fig. 3c respectively. As Fig. 3c illustrates, nodes of SLG are labeled with the class of the corresponding symbol to which the stroke belongs. A dashed edge refers to segmentation information; it indicates that a pair of strokes belongs to the same symbol. In this case, the edge label is the same as the common symbol label. On the other hand, the non-dashed edges define spatial relationships between nodes and are labeled with one of the six different possible relationships between symbols. As a consequence, all strokes belonging to the same symbol are fully connected, nodes and edges sharing the same symbol label; when two symbols are in relation, all strokes from the source symbol

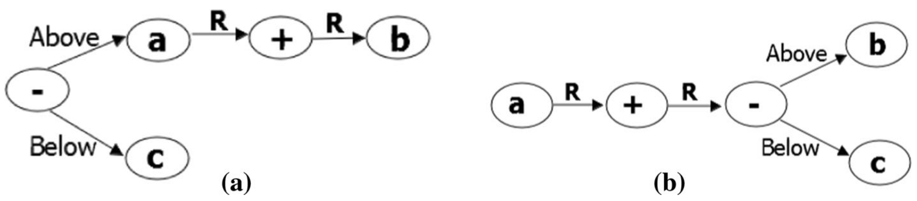


Fig. 2 The symbol relation tree (SRT) for **a** $\frac{a+b}{c}$, **b** $a+\frac{b}{c}$. '*R*' refers to *Right* relationship

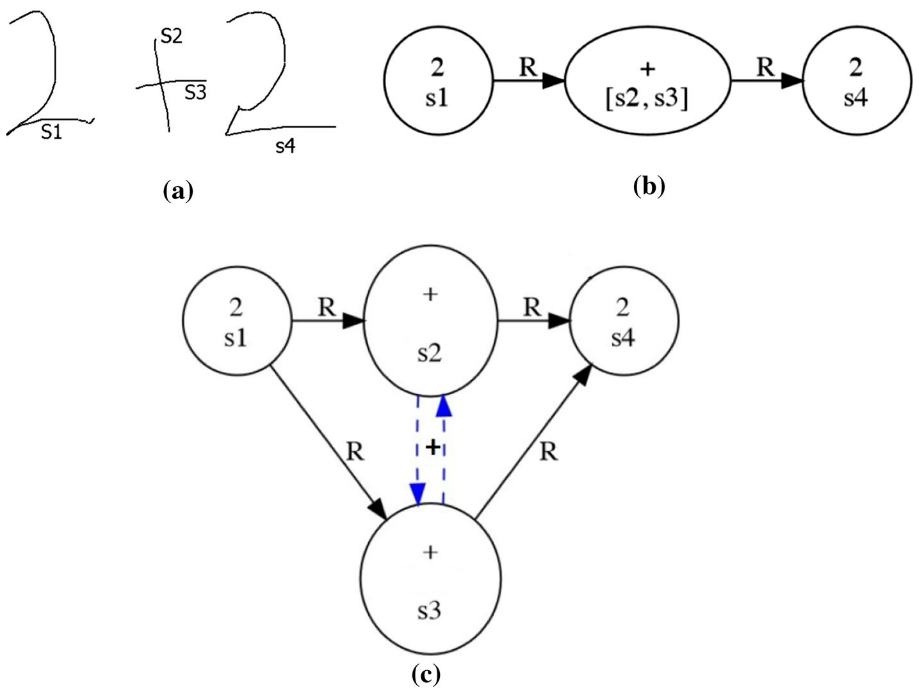


Fig. 3 **a** $2+2$ written with four strokes, **b** the symbol relation tree of $2+2$, **c** the SLG of $2+2$. The four strokes are indicated as $s1$, $s2$, $s3$, $s4$ in writing order. '*R*' is for left-right relationship

are connected to all strokes from the target symbol by edges sharing the same relationship label.

Since CROHME 2013, SLG has been used to represent mathematical expressions [4]. As the official format to represent the ground truth of handwritten math expressions and also for the recognition outputs, it allows detailed error analyses on stroke, symbol and expression levels. The recognition system that we propose works directly with the raw stroke inputs. It will produce a SLG directly comparable to the ground truth SLG. To that end, a label will be assigned to each stroke and to each pair of strokes involved in a symbol relation.

2.3 Long short-term memory networks

Recurrent neural networks (RNNs) RNNs can access contextual information and therefore are suitable for sequence labeling tasks [42]. We show an unfolded single-directional recurrent network in Fig. 4, where each node at a single time step represents a layer of network units. The network output at step t_i depends on both the current input at step t_i and the hidden state of t_{i-1}. The same weights ($w1$, $w2$, $w3$) are reused at every time step.

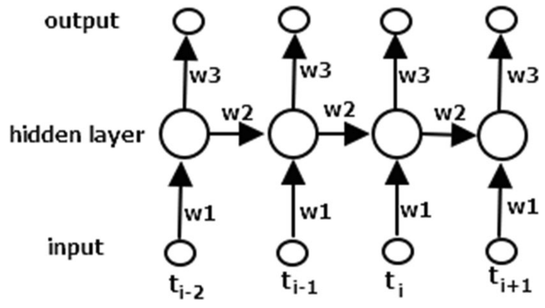


Fig. 4 An unfolded single-directional recurrent network

LSTM Unfortunately, with standard RNN architectures, the range of context that can be accessed is quite limited due to the vanishing gradient problem [43]. Long short-term memory (LSTM) [44] could address this problem by introducing a memory block which has the ability to preserve the state over long period of time. An LSTM network is similar to a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks. Each block contains one or more self-connected memory cells and three multiplicative units (the input, output and forget gates). The three gates collect activation from inside and outside the block and control the activation of the cell via multiplications. The input and output gates multiply the input and output of the cell while the forget gate multiplies the cells previous state. The only output from the block to the rest of the network emanates from the output gate multiplication.

BLSTM LSTM network processes the input sequence from past to future while Bidirectional LSTM [45], consisting of 2 separated LSTM layers, models the sequence from two opposite directions (past to future and future to past) in parallel. Both of 2 LSTM layers are connected to the same output layer. With this setup, complete long-term past and future context is available at each time step for the output layer.

Deep BLSTM DBLSTM [46] can be created by stacking multiple BLSTM layers on top of each other in order to get higher level representation of the input data. The outputs of 2 opposite hidden layer at one level are concatenated and used as the input to the next level.

Non-chain-structured LSTM A limitation of the classical LSTM network topology is that they only allow for sequential information propagation since the cell contains a single recurrent connection (modulated by a single forget gate) to its own previous value. Recently, research on LSTM has been beyond sequential structure. The one-dimensional LSTM was extended to n dimensions by introducing n recurrent connections (one for each of the cell's previous states along n dimensions) with n forget gates such that the new model could take into account the context from n sources. It is named Multidimensional LSTM (MDLSTM) dedicated to the graph structure of an n-dimensional grid such as images [42]. MDLSTM model exhibits great performances on offline handwriting recognition tasks where the input is an image [47–51].

In [8], the basic LSTM architecture was extend to tree structures for improving semantic representations. Two extensions, the Child-sum Tree-LSTM and the N-ary Tree-LSTM, were proposed to allow for richer network topology where each unit is able to incorporate information from multiple child units. The Child-sum Tree-LSTM unit conditions its components on the sum of child hidden states. It is well-suited for trees with high branching factor or whose children are unordered. The N-ary Tree-LSTM can be used on tree structures where the branching factor is at most N and where children are ordered. In parallel to the work in [8, 9] explored the similar idea and proposed *S*-LSTM model which provides a principled way of considering long-distance interaction over hierarchies, e.g., language or image parse structures. Furthermore, the DAG-structured LSTM was proposed for semantic compositionality in [10], possessing the ability to incorporate external semantics including non-compositional or holistically learned semantics.

Similar to the above-mentioned works, we will extend the chain-structured BLSTM to tree-based BLSTM, and apply this new network model for online math expression recognition.

Connectionist temporal classification (CTC) With memory capability, RNNs are suitable for the sequence labeling tasks where the context is quite important. However, to apply this recurrent network for sequence labeling, there is a troublesome problem to be solved, being that at least a loss function should be defined for the training process. When using the typical frame-wise training method, the ground truth label of each time step is required to calculate the loss function, which implies that the training data should be segmented beforehand. The network is trained to make correct label prediction at each point. However, either the pre-segmentation or making label prediction at each point, both are large burdens to users or networks.

The technique of CTC was proposed to release these two burdens. It is specifically designed for sequence labeling problems where the alignment between the inputs and the target labels is unknown. By introducing an additional 'blank' class, CTC allows the network to make label predictions at some points instead of each point in the input sequence, so long as the overall sequence of character labels is correct. As described, CTC outputs directly sequences of labels, providing no information regarding the alignment between the inputs and the target labels.

Our proposal, building the SLG from the input handwritten ME, requires a label decision for each stroke and each stroke pair used in a symbol relation. To achieve this, we extend CTC to local CTC to relatively constrain the outputs and at the same time benefit from introducing an additional 'blank' class.

3 The proposed tree-based BLSTM

This section will be focused on Tree-based BLSTM. Different with the tree structures depicted in [8, 9], we devote it to the kind of structures presented in Fig. 5 where most nodes have only one next node. In fact, this kind of structure could be regarded as several chains with shared or overlapped segments. Traditional BLSTM processes a sequence both from left to right and from right to left in order to access information coming from two directions. In our case, the tree will be processed from root to leaves and from leaves to root in order to visit all the surround context.

From root to leaves There are 2 special nodes (red) having more than one next node in Fig. 5. We name them *Mul-next node*. The hidden states of *Mul-next node* will be propagated to its next nodes equally. The forward propagation of a *Mul-next node* is the same as for a chain LSTM

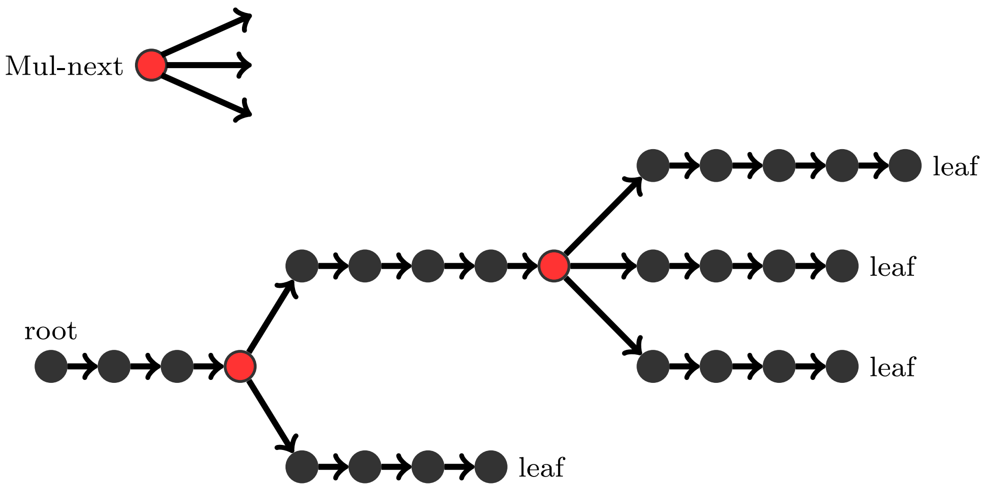


Fig. 5 A tree-based structure for chains (from root to leaves)

node; with regard to the error propagation, the errors coming from all the next nodes will be summed up and propagated to *Mul-next node*.

From leaves to root Suppose all the arrows in Fig. 5 are reversed, we have the new structure which is actually beyond a tree in Fig. 6. The 2 red nodes are still special cases because they have more than one previous nodes. We call them *Mul-previous nodes*. The information from all the previous nodes will be summed up and propagated to the *Mul-previous node*; the error propagation is processed like for a typical LSTM node.

We give the specific formulas below regarding to the forward propagation of *Mul-previous node* and the error back-propagation of *Mul-next node*. The same notations as in [42] are used here. The network input to unit i at node n is denoted a_i^n and the activation of unit i at node n is b_i^n. w_{ij} is the weight of the connection from unit i to unit j. Considering a network with I input units, K output units and H hidden units, let the subscripts ς, ϕ, ω refer to the input, forget and output gate. The subscript c refers to one of the C cells. Thus, the peephole weights from cell c to the input, forget, output gates can be denoted as $w_{c\varsigma}$, $w_{c\phi}$, $w_{c\omega}$. s_c^n is the state of cell c at node n. f is the activation function of the gates, and g and h are respectively the cell input and output activation functions. L is the loss function used for training.

We only give the equations for a single memory block. For multiple blocks the calculations are simply repeated for each block. Let $\Pr(n)$ denote the set of previous nodes of node n and $\mathrm{Ne}(n)$ denote the set of next nodes. We highlight the different parts with box compared to the classical LSTM formulas.

The forward propagation of *Mul-previous node*

Input gate

The input gate collects the activations from the current input x_i^n, the hidden unit activation b_h^p of $\Pr(n)$ and the cell state s_c^p of $\Pr(n)$. f denote the gate activation function, usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open).

$$a_\varsigma^n = \sum_{i=1}^{I} w_{i\varsigma} x_i^n + \sum_{h=1}^{H} w_{h\varsigma} \boxed{\sum_{p=1}^{|\Pr(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\varsigma} \boxed{\sum_{p=1}^{|\Pr(n)|} s_c^p} \quad (1)$$

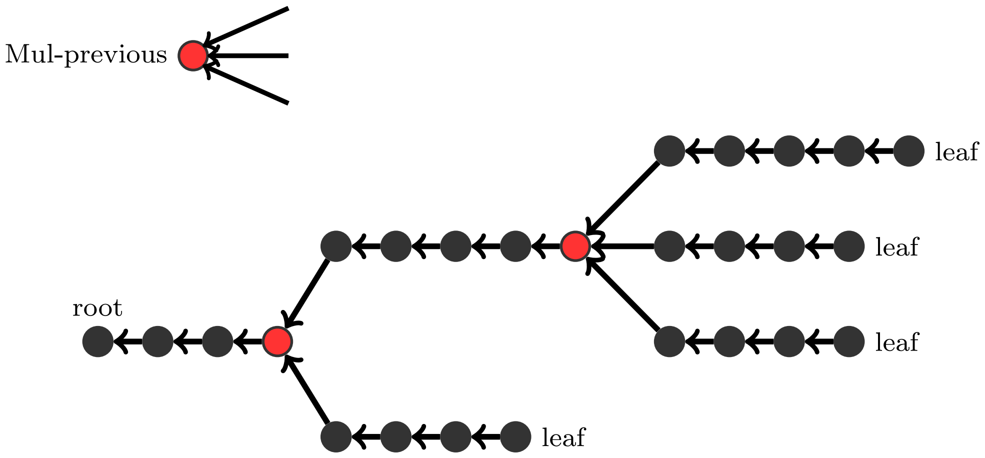


Fig. 6 A tree-based structure for chains (from leaves to root)

$$b_\varsigma^n = f(a_\varsigma^n) \tag{2}$$

Forget gate

The forget gate works in the say way as the input gate.

$$a_\phi^n = \sum_{i=1}^{I} w_{i\phi} x_i^n + \sum_{h=1}^{H} w_{h\phi} \boxed{\sum_{p=1}^{|\Pr(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\phi} \boxed{\sum_{p=1}^{|\Pr(n)|} s_c^p} \tag{3}$$

$$b_\phi^n = f(a_\phi^n) \tag{4}$$

Cell

The input to the cell includes the current input x_i^n, the hidden unit activation of $\Pr(n)$. g is the cell input activation function. The input gate controls the input of the cell via multiplication. Similarly, the forget gate decides on to which extent the cell could access the cell state s_c^p of $\Pr(n)$.

$$a_c^n = \sum_{i=1}^{I} w_{ic} x_i^n + \sum_{h=1}^{H} w_{hc} \boxed{\sum_{p=1}^{|\Pr(n)|} b_h^p} \tag{5}$$

$$s_c^n = b_\phi^n \boxed{\sum_{p=1}^{|\Pr(n)|} s_c^p} + b_\varsigma^n g(a_c^n) \tag{6}$$

Output gate

The output gate receives the activations from the current input x_i^n, the hidden unit activation of $\Pr(n)$ and the current cell state s_c^n.

$$a_\omega^n = \sum_{i=1}^{I} w_{i\omega} x_i^n + \sum_{h=1}^{H} w_{h\omega} \boxed{\sum_{p=1}^{|\Pr(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\omega} s_c^n \tag{7}$$

$$b_\omega^n = f(a_\omega^n) \tag{8}$$

Cell output

The output gate controls the output of the cell via multiplication.

$$b_c^n = b_\omega^n h(s_c^n) \tag{9}$$

The error back-propagation of *Mul-next node*

Definitions

$$\epsilon_c^n = \frac{\partial L}{\partial b_c^n} \qquad \epsilon_s^n = \frac{\partial L}{\partial s_c^n} \qquad \delta_i^n = \frac{\partial L}{\partial a_i^n} \tag{10}$$

Cell output

$$\epsilon_c^n = \sum_{k=1}^{K} w_{ck} \delta_k^n + \sum_{g=1}^{G} w_{cg} \boxed{\sum_{e}^{|\mathrm{Ne}(n)|} \delta_g^e} \tag{11}$$

ϵ_c^n refers to the partial derivative of the loss function L with respect to the output of cell c at node n.

Output gate

$$\delta_w^n = f'(a_w^n) \sum_{c=1}^{C} h(s_c^n) \epsilon_c^n \tag{12}$$

δ_w^n denotes the partial derivative of the loss function L with respect to the input of output gate at node n.

State

$$\begin{aligned} \epsilon_s^n = {} & b_w^n h'(s_c^n) \epsilon_c^n + \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} b_\phi^e \textstyle\sum_{e=1}^{|\mathrm{Ne}(n)|} \epsilon_s^e} \\ & + w_{c\varsigma} \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} \delta_\varsigma^e} + w_{c\phi} \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} \delta_\phi^e} + w_{c\omega} \delta_\omega^n \end{aligned} \tag{13}$$

ϵ_s^n represents the partial derivative of the loss function L with respect to the cell state at node n.

Cell

$$\delta_c^n = b_\varsigma^n g'(a_c^n) \epsilon_s^n \tag{14}$$

δ_c^n is the partial derivative of the loss function L with respect to the cell input at node n.

Forget gate

$$\delta_\phi^n = f'(a_\phi^n) \sum_{c=1}^{C} \boxed{\sum_{p=1}^{|\Pr(n)|} s_c^p \epsilon_s^n} \tag{15}$$

δ_ϕ^n denotes the partial derivative of the loss function L with respect to the input of forget gate at node n.

Input gate

$$\delta_\varsigma^n = f'(a_\varsigma^n) \sum_{c=1}^{C} g(a_c^n) \epsilon_s^n \tag{16}$$

δ_ς^n refers to the partial derivative of the loss function L with respect to the input of input gate at node n.

4 The tree-BLSTM-based recognition system

In this section, we elaborate orderly each stage involved in the proposed system. The input data are available as a sequence of strokes S from which we would like to obtain the correct SLG graph describing unambiguously the ME. Let $S = (s_0, \ldots, s_{n-1})$, where we assume s_i has been written before s_j for $i < j$. Algorithm 1 presents the framework of the recognition system in an algorithm procedure format for clarity.

Algorithm 1 Framework of the proposed system

```
Input:
  A sequence of strokes $S = (s_0, ..., s_{n-1})$
Output:
  SLG describing unambiguously the ME
1: Derive from $S$ an intermediate graph $G$;
2: Derive trees from $G$;
3: Label trees with tree-based BLSTM;
4: Merge labeled trees to build a SLG;
```

4.1 Derivation of an intermediate graph *G*

In a first step, we will derive from S an intermediate graph G where each node represents a stroke and edges are added according to several defined criteria. We provide several definitions related to the graph building first.

Definition 1 A stroke s_i is considered visible from stroke s_j if the straight line between their closest points does not intersect any other stroke s_k of the ink.

For example, s_1 and s_3 can see each other because the straight line between their closest points does not intersect any other stroke of the ink, here s_2 or s_4 as shown in Fig. 7. This definition is the same as the one used in [52].

Definition 2 For each stroke s_i, we define 5 regions ($R1$, $R2$, $R3$, $R4$, $R5$ shown in Fig. 8) based on it. The center of the bounding box of stroke s_i is taken as the reference point (0, 0).

The purpose of defining these 5 regions is to look for the potential *Right*, *Inside*, *Above*, *Below*, *Supscript* and *Subscript* relationships between strokes. If the center of bounding box of s_j is located in one of the five regions of stroke s_i, for example $R1$ region, we say s_j is in the $R1$ direction of s_i. A wider searching range is defined for both $R3$ and $R4$ regions. That is because in some expressions like $\frac{a+b+c}{d+e+f}$, a larger searching range means more possibilities to catch the *Above* relationship from '−' to 'a' and the *Below* relationship from '−' to 'd'.

Definition 3 Let G be a directed graph in which each node corresponds to a stroke and edges are added according to the following criteria in succession.

We defined for each stroke s_i (i from 0 to $n-2$):

- the set of strokes $S_{\text{int}}(i) = \{s_{\text{int1}}, s_{\text{int2}}, \ldots\}$, each element is from $\{s_{i+1}, \ldots, s_{n-1}\}$, and intersects s_i

For stroke s_i (i from 0 to $n-1$):

- the set $S_{\text{vis}}(i)$ of the visible leftmost (considering the center of bounding box only) strokes in five directions respectively.

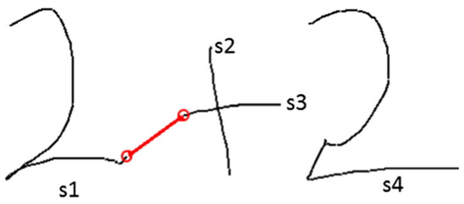


Fig. 7 Illustration of visibility between a pair of strokes. s_1 and s_3 are visible to each other

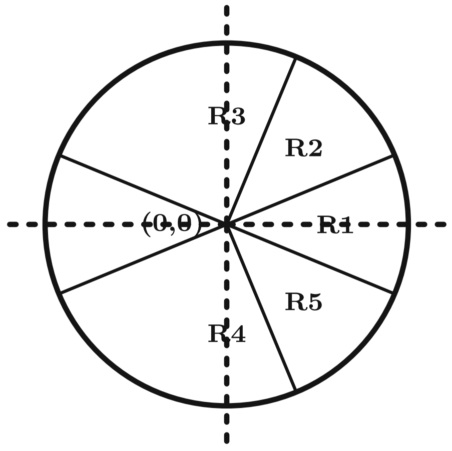


Fig. 8 Five regions for a stroke s_i. Point (0, 0) is the center of bounding box of s_i. The angle range of $R1$ region is $[-\frac{\pi}{8}, \frac{\pi}{8}]$; $R2: (\frac{\pi}{8}, \frac{3*\pi}{8}]$; $R3: (\frac{3*\pi}{8}, \frac{7*\pi}{8}]$; $R4: [-\frac{7*\pi}{8}, -\frac{3*\pi}{8})$; $R5: [-\frac{3*\pi}{8}, -\frac{\pi}{8})$

Edges from s_i to the $S_{\text{int}}(i) \bigcup S_{\text{vis}}(i)$will be added to G. Then, we check if the edge from s_i to s_{i+1} (i from 0 to $n-2$) exists in G. If not, this edge is added to G to ensure that the path covering the sequence of strokes in the time order is included in G. Each edge is tagged depending on the specific criterion we used to find it before. Consequently, we have at most 7 types of edges (*Intersection*, $R1$, $R2$, $R3$, $R4$, $R5$ and *Time*) in the graph. For those edges from s_i to the $S_{\text{int}}(i) \cap S_{\text{vis}}(i)$, the type *Intersection* is assigned.

Figure 9 illustrates the process of deriving graph from raw input step by step using the example of $\frac{f}{a} = \frac{b}{f}$. First according to the 10 strokes in the raw input (Fig. 9a), we create 10 nodes, one for each stroke; for each stroke, look for its intersecting strokes and add the corresponding edges labeled with *Intersection* between nodes (Fig. 9b); proceeding to next step, for each stroke, look for its the visible rightmost strokes in five directions respectively and add the corresponding edges labeled as one of $R1$, $R2$, $R3$, $R4$, $R5$ between nodes if the edges do not exist in the graph (Fig. 9c); finally, check if the edge from s_i to s_{i+1} (i from 0 to $n-2$) exists in G and if not, add this edge to G labeled as *Time* to ensure that the path covering the sequence of strokes in the time order is included in G (Fig. 9d).

4.2 Derivation of trees from *G*

As described before in Sect. 4.1, we derive a graph from the raw input considering the temporal and spatial information. Then we will try to label nodes and edges of G correctly in order to build a SLG finally. The solution proposed in this work is to derive multiple trees from G, then recognize the trees using the tree-based BLSTM model.

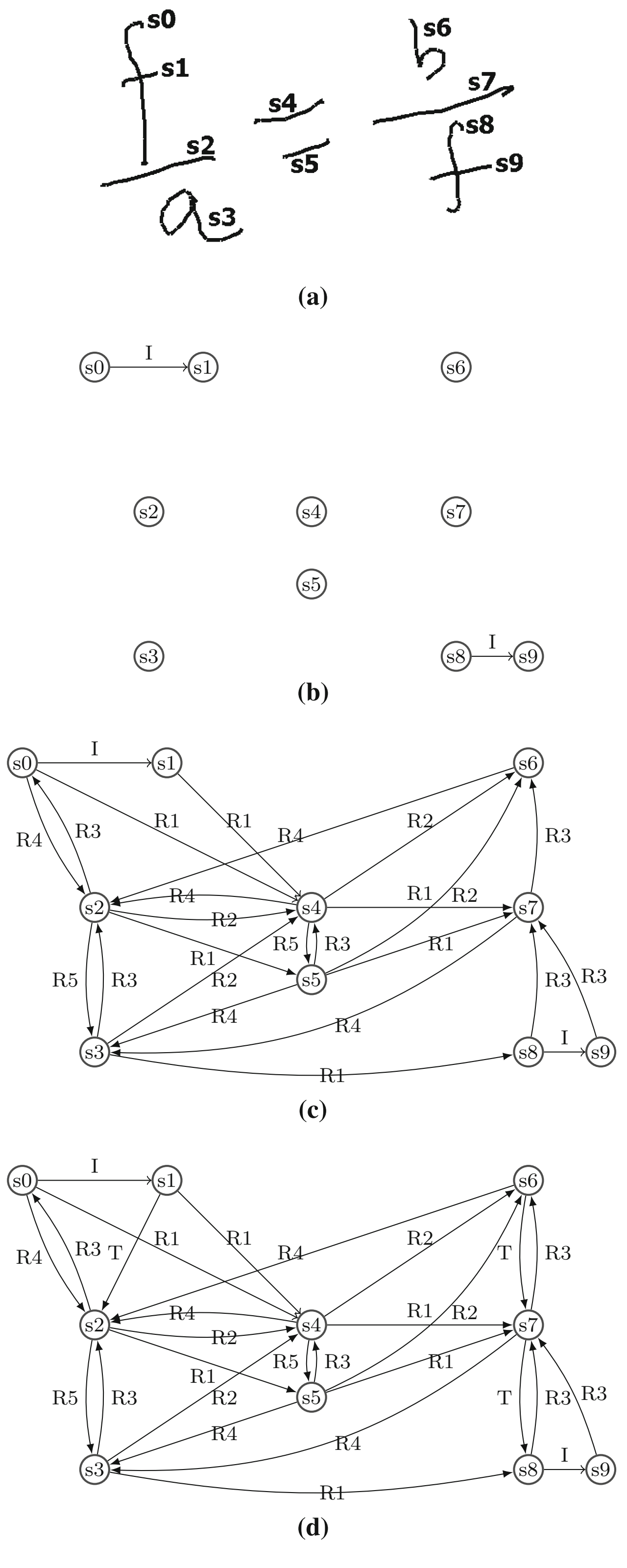


Fig. 9 **a** $\frac{f}{a} = \frac{b}{f}$ is written with 10 strokes, **b** add *Intersection* edges; **c** add *R*1, *R*2, *R*3, *R*4, *R*5 edges, **d** add *Time* edges. *I* : *Intersection*, *T* : *Time*

There exist different strategies to derive trees from *G*. In any of the cases, a start node should be selected first. We take the leftmost (considering the leftmost point in a stroke) stroke as the starter. From the starting node, we traverse the graph with the Depth-First Search algorithm. Each node should be visited only once. When there are more than one edge outputting from one node, the visiting order will follow the sequence of (*Intersection*, *R*1, *R*3, *R*4, *R*2, *R*5, *Time*). With this strategy, a tree is derived to which we give the name *Tree-Left-R1*. It is dedicated to catch *R*1 relationship. Figure 10 illustrates the derived *Tree-Left-R1* for the ME $\frac{f}{a} = \frac{b}{f}$ written with 10 strokes. The *Intersection* edge is on the top of list, and it is because we assume that a pair of intersecting strokes belongs to a single symbol. In Fig. 10, *Tree-Left-R1* is depicted in red with the root in *s*2. Note that in this case, all the nodes are accessible from the start node *s*2. However, as *G* is a directed graph, some nodes are not reachable from one starter in some cases. Therefore, we consider deriving trees from different starters. Besides the leftmost stroke, it is interesting to derive trees from the first input stroke *s*0 since sometimes users start writing an expression from its root. Note that in some cases, the leftmost stroke and stroke *s*0 could be the same one. We replace the left-most stroke with stroke *s*0 and keep the same strategy to derive the tree. The new tree is named as *Tree-0-R1*.

Finally, if *s*0 is taken as the starting point and time order is considered first, a special tree is obtained which we call *Tree-Time*. *Tree-Time* is proposed with the aim of having a good cover of segmentation edges since users usually write a multi-stroke symbol continuously. As a matter of fact, it is a chain structure. *Tree-Time* is defined by $s0 \rightarrow s1 \rightarrow s2 \rightarrow s3 \ldots \rightarrow s9$ for the expression in Fig. 10. Table 1 offers a clear summary of the 3 derived trees from the graph. The experiment results presented in Sect. 5.2.2 prove the effectiveness of the method for deriving trees.

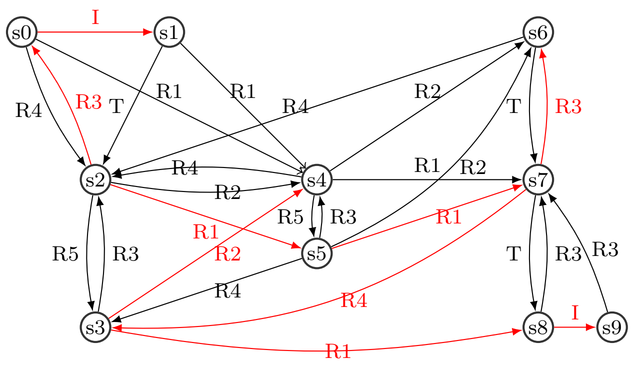


Fig. 10 The derived graph of the handwritten ME $\frac{f}{a} = \frac{b}{f}$ shown in Fig. 9a. *Tree-Left-R1* is highlighted with red color. *s*2 is the root of it. *I* : *Intersection*, *T* : *Time* (color figure online)

4.3 Feed the inputs of the tree-based BLSTM

In Sect. 4.2, we derived trees from the intermediate graph. Nodes of the tree represent visible strokes and edges denote the relationships between pairs of strokes. We would like to label each node and edge correctly with the Tree-based BLSTM model, aiming to build a complete SLG finally. To realize this, the first step is to feed the derived tree into the Tree-based BLSTM model.

The solution is to go from the previous trees defined at the stroke level down to a tree at the point level, points being the raw information that are recorded along the pen trajectory in the online signal. To be independent from the different writing speeds, an additional re-sampling process should be carried out with a fixed spatial step. In the considered trees, nodes, which represent strokes, are re-sampled with a fixed spatial step, and the same holds for edges by considering the straight lines in the air between the last point and the first point of a pair of strokes that are connected in the tree. This is illustrated in Fig. 11, where the re-sampled points are displayed inside the nodes (on-paper points for node) and above the edges (in-air points for edge). Since this tree will be processed by the BLSTM network, we need for the training stage to assign it a corresponding ground truth. We derive it from the SLG by using the corresponding symbol label of the strokes (nodes) for the on-paper points and the corresponding symbol or relationship label for the in-air points (edges) when this edge exists in the SLG. When an edge of the tree does not exist in the SLG, the label *NoRelation* noted '_' will be used. In this way, an edge in the graph which was originally denoted with a I, Ri $(i = 1\ldots5)$ or T relation will be assigned with one of the 7 labels: $(Right, Above, Below, Inside, Superscript, Subscript, \_)$ or a symbol label when the two strokes are belonging to the same symbol. Totally, for the ground truth, we have 108 classes(101 symbol classes + 6 relationships + *NoRelation*).

The number of re-sampling points depends on the scale of the strokes with regard to the full expression scale. For each node or edge, we re-sample with $10 \times l/d$ points. Here, l refers to the length of a visible stroke or a straight line connecting 2 strokes and d refers to the average diagonal of the bounding boxes of all the strokes in an expression. Subsequently, for every point $p(x, y)$ we

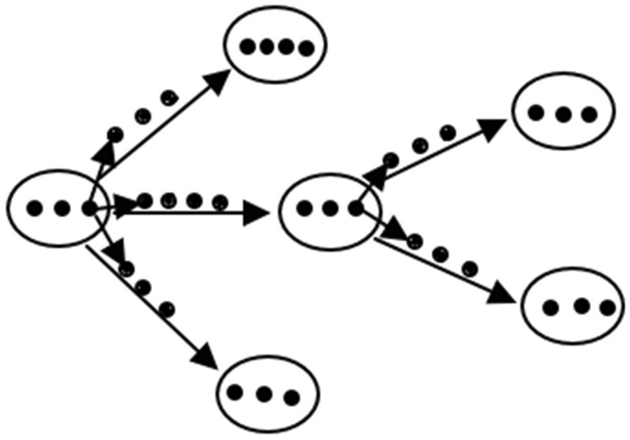


Fig. 11 A re-sampled tree. The small arrows between points provide the directions of information flows. With regard to the sequence of points inside one node or edge, most of small arrows are omitted

compute 5 features [sinθ, cosθ, sinϕ, cosϕ, PenUD]. The full description of the features can be found in [12].

4.4 Training process

Figure 12 illustrates a tree-based BLSTM network with one hidden level. To provide a clear view, we only draw the full network on a short sequence (red) instead of a whole tree. Globally, the data structure we are dealing with is a tree; locally, it consists of several short sequences. For example, the tree presented in Fig. 12 has 6 short sequences one of which is highlighted with red color. The system processes each node or edge (which is a short sequence in fact) separately but following the order with which the correct propagation of activation or errors could be ensured.

The training process of a short sequence (the red one in Fig. 12 for example) is similar to the classical BLSTM model except that some information from adjacent short sequences should be taken into account. In the classical BLSTM case, the incoming activation or error of a short sequence is initialized as 0.

Forward pass Here, when proceeding with the forward pass from the input layer to the output layer, for the hidden layer (from root to leaves), we need to consider the coming information from the root direction and for the hidden layer (from leaves to root), we need to consider the coming information from the leaves direction. Obviously, no matter which kind of order for processing sequence we are

Table 1 The different types of derived trees

Type	Root	Traverse algorithm	Visiting order
Tree-Left-R1	The leftmost stroke	Depth-First Search	(*Intersection*, *R*1, *R*3, *R*4, *R*2, *R*5, *Time*)
Tree-0-R1	*s*0	Depth-First Search	(*Intersection*, *R*1, *R*3, *R*4, *R*2, *R*5, *Time*)
Tree-Time	*s*0	Depth-First Search	Only the time order

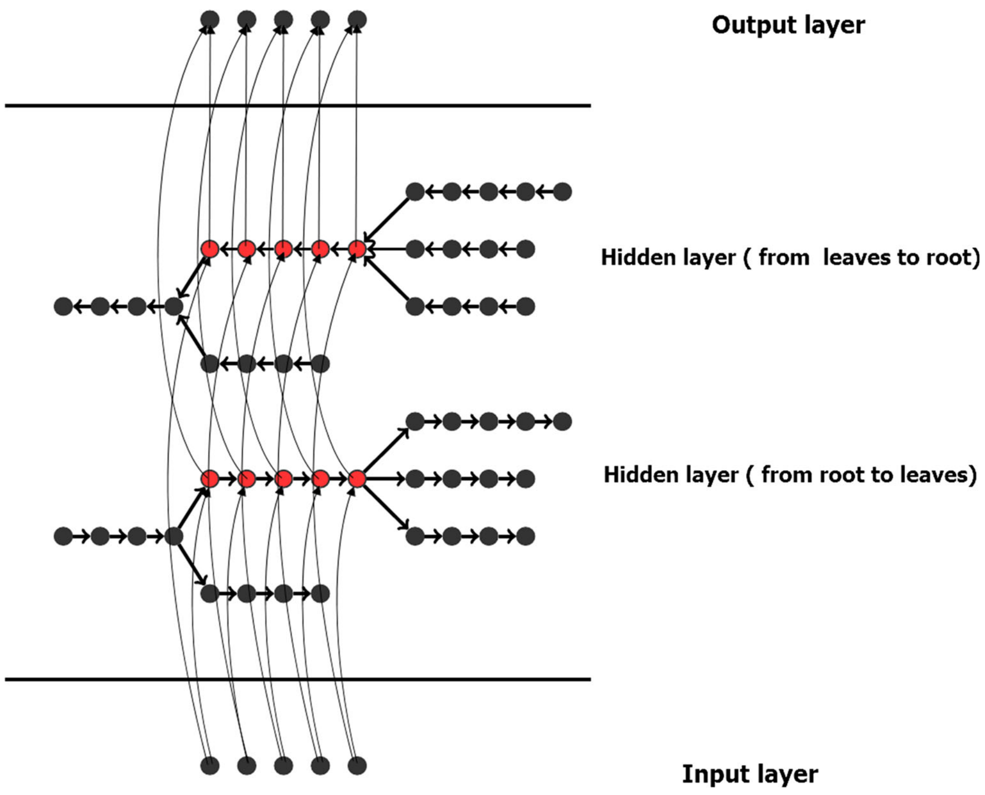

Output layer
Hidden layer (from leaves to root)
Hidden layer (from root to leaves)
Input layer

Fig. 12 A tree-based BLSTM network with one hidden level. We only draw the full connection on one short sequence (red) for a clear view (color figure online)

following, it is not possible to have the information from both directions in one run. Thus another stage which we call *pre-computation* is required. The pre-computation stage has two runs: (1) From the input layer to the hidden layer (from root to leaves), we process the short sequence consisting of the root point first and then the next sequences. In this run, each sequence in the tree stores the activation from the root direction. (2) From the input layer to the hidden layer (from leaves to root), we process the short sequences consisting of the leaf point first and then the next sequences. In this run, each sequence in the tree sums and stores the activation from the leaf direction. After pre-computation stage, the information from both directions are available to each sequence thus the forward pass from input to output is straightforward.

Error propagation The backward pass of tree-based BLSTM network has 2 parallel propagation paths: (1) one is from the output layer to hidden layer (from root to leaves), then to the input layer; (2) the other one is from the output layer to hidden layer (from leaves to root), then to the input layer. As these 2 propagation are independent, no pre-computation stage is needed here. For propagation (1), we process the short sequences consisting of the leaf point first and then the next sequences. For propagation (2), we process the short sequence consisting of the root point first and then the next sequences. Note that when there are several hidden levels in the network, a pre-computation stage is required also for error propagation.

Loss function It is known that BLSTM and CTC stage have better performance when a 'blank' label is introduced during the training process [53], so that decision can be made only at some point in the input sequence. One of the characteristics of CTC is that it does not provide the alignment between the input and output, just the overall sequence of labels. As we need to assign each stroke a label to build a SLG, a relatively precise alignment between the input and output is preferred. Thus, a local CTC algorithm is used in this work aiming to limit the label into the corresponding stroke.

Inside each short sequence, or we can say each node or edge, a local CTC loss function is easy to be computed from the output probabilities related to this short sequence. The total CTC loss function of a tree is defined as the sum of all local CTC loss functions regarding to all the short sequences in this tree.

Since each short sequence has one label, the possible labels of the points in one short sequence are shown in Fig. 13. For example, suppose character *c* is written with one stroke and 3 points are re-sampled from the stroke. The possible labels of these points can be *ccc*, *cc*–, *c*– –, – –*c*, –*cc* and –*c*– ('–' denotes 'blank'). More generally, the number of possible label sequences is $n * (n+1)/2$ (n is the number of points), which is actually 6 with the proposed example.

Given the tree input represented as X consisting of N short sequences, each short sequence could be denoted as

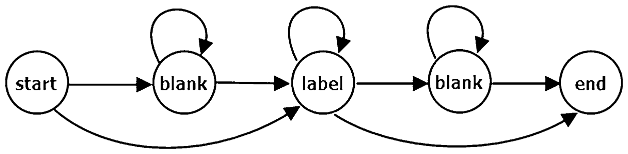


Fig. 13 The possible labels of points in one short sequence

$X_i, i = 1, \ldots, N$ with the ground truth label l_i and the length T_i. l_i' represents the label sequence with blanks added to the beginning and the end of l_i, i.e., $l_i' = (blank, l_i, blank)$ of length 3. The forward variable $\alpha_i(t, u)$ denotes the summed probability of all length t paths that are mapped by F onto the length $u/2$ prefix of l_i, where u is from 1 to 3 and t is from 1 to T_i. The mapping function F is defined as first removing the repeated labels and then the blanks (–) from the paths. For example considering an short sequence of length 6, two possible paths could be $--aaa-$, $---aa-$. The mapping function works like: $F(--aaa-) = F(---aa-) = a$.

Similarly, the backward variable $\beta_i(t, u)$ denotes the summed probabilities of all paths starting at $t+1$ that complete l_i when appended to any path contributing to $\alpha_i(t, u)$. Figure 14 demonstrates the CTC forward-backward algorithm limited in one stroke.

With the CTC forward-backward algorithm (referring to [42] for details), we can compute the $\alpha_i(t, u)$ and $\beta_i(t, u)$ for each point t and each allowed positions u at point t. The local CTC loss function $L(X_i, l_i)$ is defined as the negative log probability of correctly labeling the short sequence X_i:

$$L(X_i, l_i) = -\ln p(l_i|X_i) \tag{17}$$

According to Equation (7.26) in [42], we can rewrite $L(X_i, l_i)$ as:

$$L(X_i, l_i) = -\ln \sum_{u=1}^{3} \alpha_i(t, u)\beta_i(t, u) \tag{18}$$

Then the errors will be back propagated to the output layer, then the hidden layer, finally to the entire network. The weights in the network will be updated after each entire tree structure is processed.

The CTC loss function of a entire tree structure is defined as the sum of the errors with regards to all the short sequences in this tree:

$$L(X, l) = \sum_{i=1}^{N} L(X_i, l_i) \tag{19}$$

This formula is used for evaluating the performance of the network, and therefore could be as the metric to decide the training process stops or not.

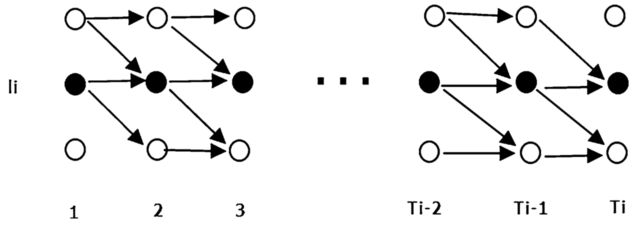


Fig. 14 CTC forward-backward algorithm in one stroke X_i. Black circle represents label l_i and white circle represents blank. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated in the reverse direction

4.5 Recognition process

As mentioned, the system treats each node or edge as a short sequence. A simple decoding method is adopted here. We choose for each node or edge the label which has the highest cumulative probability over the short sequence. Suppose that y_i^j is the probability of outputting the i label at the j point. The probability of outputting the i label can be computed as $P_i = \sum_{j=1}^{s} y_i^j$, where s is the number of points in a short sequence. The label with the highest probability is assigned to this short sequence.

4.6 Post-process

Several trees regarding to one expression will be merged to build a SLG after labeling. Besides the merging strategy, in this section, we consider several structural constraints when building the SLG. Generally, 5 steps are included in post-process:

(1) *Merge trees* Each node or edge belongs at least to one tree, but possibly to several trees. Hence, several recognition results can be available for a single node or edge. We take an intuitive and simple way to deal with the problem of multiple results, choosing the one with the highest probability.

(2) *Symbol segmentation* We look for the symbols using connected component analysis: a connected component where nodes and edges have the same label is a symbol.

(3) *Relationships* We solve two possible kinds of conflicts in this step. (a) Perhaps between two symbols, there exists edges in both directions. Then, in each direction, we choose the label having the maximum probability. If the labels in two direction are both one of (*Right*, *Above*, *Below*, *Inside*, *Superscript*, *Subscript*) as illustrated in Fig. 15a, we also choose the one having the larger probability. (b) Another type of conflict could be the case illustrated in Fig. 15b where one symbol has two (or

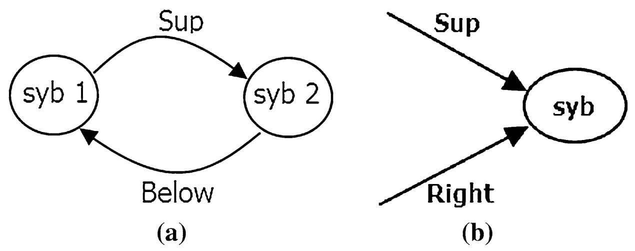


Fig. 15 Possible relationship conflicts existing in merging results

more) input relationships (one of 6 relationships). As introduced in Sect. 2.2.1, there is at most one input relationship for each node (symbol) in SRT. Therefore, when one symbol has two (or more) input relationships, we choose for it the one having the maximum probability.

(4) *Make connected SRT* As SRT is a connected tree (this is a structural constraint, not a language specific constraint), there should be one root node and one or multiple leaf nodes. Each node has only one input edge, except the root node. After performing the first three steps, we still have the possibility to output a SRT containing several root nodes, in other words, being a forest instead of a tree. To address this type of error, we take a hard decision but quite simple: for each root **r** (except the one inputted earliest), add a *Right* edge to **r** from the leaf being the one nearest to **r** considering input time. We choose *Right* since it appears most in math expressions based on the statistics.

(5) *Add edges* According to the rule that all strokes in a symbol have the same input and output edges and that double-direction edges represent the segments, some missing edges can be completed automatically.

4.7 Time complexity

In this section, we analyze the time complexity of our tree-BLSTM-based recognition system and compare it with other grammar-driven solutions. As explained before, the proposed system consists of several sequential steps for each of which we will list the time complexity individually. First of all, the time complexity of the algorithm for building an intermediate graph is $O(n^2)$. Then to derive trees, the Depth-First Search algorithm requires $O(n+e)$ time where $e = 7n$ as there are at most $7n$ edges in the intermediate graph. For activating the BLSTM, it requires $O(n)$ for nodes and $O(n)$ for edges (we are in a tree). The same time complexity holds for the recognition process. Finally, different operations in post-process take linear time also. Therefore, the time complexity of our system is $O(n^2)$ in fact. Now, we consider the grammar-driven solutions. The time complexity of grammar parsing algorithm is $O(n^3|p|)$ at least where $|p|$ denotes the number of production rules. For example, it is $O(n^3 log(n)|p|)$ for the system proposed in [35]. Thus, we have a lower time complexity, $O(n^2)$ VS $O(n^3|p|)$ (at least).

5 Experiments

Data sets The complete data set from CROHME 2014 is used, 8834 expressions for training and 983 expressions for test. We extract randomly 10% of the 8834 expressions of the training set as a validation set. To get more recent comparison with the state of the art, we have also use the last CROHME 2016 data set to evaluate the best configuration. The training data set remains the same as CROHME 2014. However, 1147 new expressions are included in CROHME 2016 test data set.

Setup We developed the tree-based BLSTM recognition system with the RNNLIB library.[1] Several types of configurations are included in this paper: *Network* (1), (2), (3), (4). The first one consists of one bidirectional hidden level (two opposite LSTM layers of 100 cells). This configuration has obtained good results in both handwritten text recognition [5] and handwritten math symbol classification [6, 54]. *Network* (2) is a deep structure with two bidirectional hidden levels, each containing two opposite LSTM layers of 100 cells. Network (3) and Network (4) have 3 bidirectional hidden levels and 4 respectively. The setup about the input layer and output layer remains the same. The size of the input layer is 5 (5 features); the size of the output layer is 109 (101 symbol classes + 6 relationships + *NoRelation* + *blank*).

With the Label Graph Evaluation library (LgEval) [55], the recognition results can be evaluated on symbol level and on expression level. We introduce several evaluation criteria: symbol segmentation (Segments), refers to a symbol that is correctly segmented whatever the label is; symbol segmentation and recognition (Seg + Class), refers to a symbol that is segmented and classified correctly; spatial relationship classification (Tree Rels.), a correct spatial relationship between two symbols requires that both symbols are correctly segmented and with the correct relationship label.

5.1 Experiment 1

Due to the deep structure, DBLSTM could generate higher level representation of the input data. In this experiment, we would like to see the effects of the depth of the network

[1] Graves A. RNNLIB: A recurrent neural network library for sequence learning problems. http://sourceforge.net/projects/rnnl/.

on the recognition results. And then according to the results, we choose the proper network configurations for the task. For each expression, the chain structure *Tree-Time* is derived to train the classifier.

The evaluation results on symbol level and global expression level are presented in Tables 2 and 3 respectively. From the tables, we can conclude that as the network turns to be deeper, the recognition rate first increases and then stays at a relatively stable level. There is a large increase from *Network (1)* to *Network (2)*, a slight increase from *Network (2)* to *Network (3)* and no improvement from *Network (3)* to *Network (4)*. These results show that 3 bidirectional hidden levels in the network is a proper option for the task in this work. Network with depth larger than 3 bring no improvement but higher computational complexity. Thus, for the coming experiments we will not take into account *Network (4)* any more.

5.2 Experiment 2

In this section, we carry out experiments merging several trees to improve the coverage of the graph. As a first try, we derive only 3 trees, *Tree-Time*, *Tree-Left-R1* and *Tree-0-R1* for each expression to train the classifiers separately. With regards to each tree, we consider 3 network configurations, being *Network (1)*, *Network (2)*, *Network (3)*. Thus, we have 9 classifiers totally in this section. After training, we use these 9 classifiers to label the relevant trees and finally merge them to build a valid SLG. We merge the 3 trees labeled by the corresponding 3 classifiers which have the same network configuration to obtain the systems (1, *Merge3*), (2, *Merge3*), (3, *Merge3*). Then we merge the 3 trees labeled by all these 9 classifiers to obtain the system *Merge 9*.

Table 2 The symbol level evaluation results on CROHME 2014 test set with *Tree-Time* only

System	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
1. *Tree-Time*	92.93	84.82	84.12	76.78	60.70	76.19
2. *Tree-Time*	95.10	90.47	87.53	83.27	65.06	83.18
3. *Tree-Time*	95.43	91.13	88.26	84.28	65.45	83.57
4. *Tree-Time*	95.57	91.21	87.81	83.80	65.98	82.85

Table 3 The expression level evaluation results on CROHME 2014 test set with *Tree-Time* only

System	Correct (%)	≤ 1 error	≤ 2 errors	≤ 3 errors
1. *Tree-Time*	12.41	20.24	26.14	30.93
2. *Tree-Time*	16.09	25.46	32.28	37.27
3. *Tree-Time*	16.80	25.56	32.89	38.09
4. *Tree-Time*	16.19	25.97	33.20	38.09

5.2.1 Intermediate graph evaluation

In Sect. 4.1, we described the method for deriving an intermediate graph from a sequence of strokes. This method should be evaluated to show its effectiveness. In [56], Hu evaluates the graph representation model by comparing the edges of the graph with ground truth edges at the stroke level where the recall and precision rates are considered both. We would like to take a similar but more intuitive method, detecting the missing and unnecessary relationships in the graph, which is at the symbol level. In order to evaluate if the selected trees will allow to recognize the symbol, we skip the recognition step in the process and label nodes and edges using the ground truth. The obtained graphs are then evaluated, with the standard with CROHME tool LgEval [55]. The symbols correctly recognized are those which could be recognized in the case of a perfect classification of each node and edge. Unrecognized symbol are due to missing edges. Obviously, a complete graph always leads to a perfect recognition.

Table 4 present the evaluation results of the intermediate graph on CROHME 2014 test set at the symbol level. As shown, the recall rate and the precision rate of segmentation task are almost 100%, telling that the intermediate graph catches almost all the segmentation edges in the ground truth SLG. With regard to relationship recognition task, the graph catches 93.99% spatial relationships in expressions, missing around 6%. Furthermore, as shown by the precision rate of the tree relations, there are in the graph more than half of the spatial relationships which are unnecessary. These unnecessary edges are expected to be labeled as *NoRelation* by the classifiers later.

5.2.2 Derived trees evaluation

As mentioned in the beginning of Sect. 5.2, we derive 3 trees, *Tree-Time*, *Tree-Left-R1* and *Tree-0-R1* for each

Table 4 Intermediate graph evaluation results at the symbol level on CROHME 2014 test set (provided the ground truth labels of nodes and edges)

Model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Graph G	99.97	99.93	99.96	99.92	93.99	43.51

expression to train the classifiers separately. Each kind of tree derived from the intermediate graph and the combination of these 3 trees should also be evaluated by the same way as the intermediate graph evaluation to check the coverage of them to capture the SLG.

As shown in Table 5, we provide the evaluation results of the derived trees (individuals and combination) on CROHME 2014 test set at the symbol level. *Tree-Time* catch 75.54% ground truth spatial relationships and 23.16% (false positive/targets) unnecessary ones. The evaluation results of *Tree-Left-R1* is very close to *Tree-0-R1* since the first input stoke is the leftmost stroke for many cases. They capture around 68% ground truth spatial relationships and around 70% unnecessary ones. When we combine these 3 trees, a better ground truth relationships coverage, 92.17%, is achieved with the unnecessary ones of 60.50%. Compared to the intermediate graph, the combination of these 3 derived trees catches slightly less ground truth relationships but greatly less unnecessary relationships which verify the effectiveness of the method for deriving trees.

5.2.3 Classifiers evaluation

In this section we now label the nodes and edges of derived trees with the BLSTM classifier. The evaluation results on symbol level and global expression level are presented in Tables 6 and 7 respectively. We give both the individual tree recognition results and the merging results (bold in Tables 6 and 7) in each table. *Tree-Time* covers all the strokes of the input expression but can miss some relational edges between strokes; *Tree-Left-R1* and *Tree-0-R1* could catch some additional edges which are not covered by *Tree-Time*. The experiment results also verified this tendency. Compared to (3, *Tree-Time*), the symbol segmentation and classification results of (3, *Merge3*) stay at almost the same level while the recall rate of relationship classification is greatly improved (about 12%). The different recognition results of network (2) are systematically increased when compared to (1) as the deep structure could get higher level representations of the input data. The performance of network (3) is moderately improved when compared to (2), just as same as the case in Experiment 1. When we consider merging all these 9 classifiers, we also get a slight improvement as shown by *Merge 9*.

We compare the results of *Merge 9* to the systems in CROHME 2014. With regard to the symbol classification and recognition rates, our system performs better than the second-ranked system in CROHME 2014. For relationship classification rate, our system reaches the level between the second-ranked and the third-ranked systems in CROHME 2014. The global expression recognition rate is 29.91%, ranking third in all the participated systems. When we compute the recognition rate with $\leq$ 3 errors, our result is 50.15%, very close to the second-ranked system (50.20%).

The top ranked system MyScript is built on the principle that segmentation, recognition and interpretation have to be handled concurrently and at the same level in order to result in the best candidate. They use a much larger training data set which is not available to the public. System València parses expressions using two-dimensional stochastic context-free grammars. It is an advanced model

Table 5 The derived trees evaluation results at the symbol level on CROHME 2014 test set (provided the ground truth labels of nodes and edges)

Model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Tree-Time	99.73	99.45	99.72	99.44	75.54	76.54
Tree-Left-R1	94.07	87.85	94.06	87.85	68.86	49.91
Tree-0-R1	94.51	88.16	94.50	88.15	67.98	49.03
Merge3	99.84	98.74	99.83	98.73	92.17	60.37

Table 6 The symbol level evaluation results on CROHME 2014 test set with 3 trees, including the experiment results in this work and CROHME 2014 participant results

System	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
1. *Tree-Time*	92.93	84.82	84.12	76.78	60.70	76.19
1. *Tree-Left-R1*	84.82	72.49	72.80	62.21	44.34	57.78
1. *Tree-0-R1*	85.31	72.88	74.17	63.37	42.92	60.08
1. *Merge3*	**93.53**	**87.20**	**86.10**	**80.28**	**71.16**	**66.13**
2. *Tree-Time*	95.10	90.47	87.53	83.27	65.06	83.18
2. *Tree-Left-R1*	86.71	75.64	76.85	67.03	48.14	61.91
2. *Tree-0-R1*	87.52	76.66	77.00	67.45	48.14	63.04
2. *Merge3*	**95.01**	**90.05**	**88.38**	**83.76**	**76.20**	**72.28**
3. *Tree-Time*	95.43	91.13	88.26	84.28	65.45	83.57
3. *Tree-Left-R1*	88.03	78.13	78.56	69.72	50.31	65.87
3. *Tree-0-R1*	87.41	77.02	77.63	68.40	48.23	64.28
3. *Merge3*	**95.25**	**90.70**	**88.90**	**84.65**	**77.33**	**73.72**
Merge 9	**95.52**	**91.31**	**89.55**	**85.60**	**78.08**	**74.64**
	CROHME 2014 participant results					
MyScript	98.42	98.13	93.91	93.63	94.26	94.01
València	93.31	90.72	86.59	84.18	84.23	81.96
Nantes	89.43	86.13	76.53	73.71	71.77	71.65
RIT-CIS	88.23	84.20	78.45	74.87	61.38	72.70
RIT-DRPL	85.52	86.09	76.64	77.15	70.78	71.51
Tokyo	83.05	85.36	69.72	71.66	66.83	74.81
São Paulo	76.63	80.28	66.97	70.16	60.31	63.74

Table 7 The expression level evaluation results on CROHME 2014 test set with 3 trees, including the experiment results in this work and CROHME 2014 participant results

System	Correct (%)	≤ 1 error	≤ 2 errors	≤ 3 errors
i, *Tree-Time*	12.41	20.24	26.14	30.93
i, *Tree-Left-R1*	5.9	10.58	15.99	19.94
i, *Tree-0-R1*	5.39	10.47	16.28	20.14
i, *Merge3*	**19.94**	**27.57**	**33.88**	**39.37**
ii, *Tree-Time*	16.09	25.46	32.28	37.27
ii, *Tree-Left-R1*	6.82	13.33	20.14	23.19
ii, *Tree-0-R1*	6.41	13.02	18.41	23.40
ii, *Merge3*	**25.94**	**36.72**	**42.32**	**46.59**
iii, *Tree-Time*	16.80	25.56	32.89	38.09
iii, *Tree-Left-R1*	8.55	15.26	20.96	24.52
iii, *Tree-0-R1*	7.93	13.63	19.63	25.43
iii, *Merge3*	**29.30**	**39.06**	**43.64**	**48.02**
Merge 9	**29.91**	**39.94**	**44.96**	**50.15**
	CROHME 2014 participant results			
MyScript	62.68	72.31	75.15	76.88
València	37.22	44.22	47.26	50.20
Nantes	26.06	33.87	38.54	39.96
Tokyo	25.66	33.16	35.90	37.32
RIT-DRPL	18.97	28.19	32.35	33.37
RIT-CIS	18.97	26.37	30.83	32.96
São Paulo	15.01	22.31	26.57	27.69

Table 8 The symbol level evaluation results on CROHME 2016 test set with the system of *Merge 9*, along with CROHME 2016 participant results

System	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Merge 9	95.64	91.44	89.84	85.90	77.23	74.08
	CROHME 2016 participant results					
MyScript	98.89	98.95	95.47	95.53	95.11	95.11
Wiris	96.49	97.09	90.75	91.31	90.17	90.79
Tokyo	91.62	93.25	86.05	87.58	82.11	83.64
São Paulo	92.91	95.01	86.31	88.26	81.48	84.16
Nantes	94.45	89.29	87.19	82.42	73.20	68.72

Table 9 The expression level evaluation results on CROHME 2016 test set with the system of *Merge 9*, along with CROHME 2016 participant results

System	Correct (%)	≤ 1 error	≤ 2 errors
Merge 9	27.03	35.48	42.46
	CROHME 2016 participant results		
MyScript	67.65	75.59	79.86
Wiris	49.61	60.42	64.69
Tokyo	43.94	50.91	53.70
São Paulo	33.39	43.50	49.17
Nantes	13.34	21.02	28.33

evolution of the system presented in [34]. System Nantes simultaneously optimizes expression segmentation, symbol recognition, and 2-D structure recognition under the restriction of an expression grammar [33]. The approach transforms the recognition problem into a search for the best possible interpretation of a sequence of input strokes. Furthermore, all the top 4 systems in the CROHME 2014 competition are grammar-driven solutions which need a large amount of manual work and a high computational complexity. There is no grammar considered in our system.

We also evaluate the system of *Merge 9* on CROHME 2016 test data set (Tables 8, 9). As can be seen in Table 8, compared to other participated systems in CROHME 2016, our system is still competitive on symbol segmentation and classification task. For relationship recognition task, there is room for improvement. The results at expression level are presented in Table 9. The global expression recognition rate is 27.03%.

5.3 Error analysis

In this section, we make a deep error analysis of the recognition result of (*Merge 9*) to better understand the system and to explore the directions for improving recognition rate in future. The Label Graph Evaluation library (LgEval) [55] evaluates the recognition system by comparing the output SLG of each expression with its ground truth SLG. Thus, node label confusion matrix and edge label confusion matrix are available. Based on the two confusion matrices, we analyze the errors specifically below.

Node label In table 10, we list the types of SLG node label error which have a high frequency on CROHME 2014 test set recognized by (*Merge 9*) system. The first column gives the outputted node labels by the classifier; the second column provide the ground truth node labels, along with the number of nodes with each label; the last column records the corresponding number of occurrences, also the percentages. As can be seen from the table, the most frequent error ($x \rightarrow X$, 46) belongs to the type of the lowercase-uppercase errors. Moreover, ($p \rightarrow P$, 24), ($c \rightarrow C$, 16), ($X \rightarrow x$, 16) and ($y \rightarrow Y$, 14) also belong to the same type of lowercase-uppercase errors. Another type of error which happens quite often in our experiment is the similar-look error, such as ($x \rightarrow \times$, 26), ($\times \rightarrow x$, 10), ($z \rightarrow 2$, 10), ($q \rightarrow 9$, 10) and so on. With more training samples, we can expect a better discrimination of these similar classes.

Table 10 Illustration of node (SLG) label errors of (*Merge 9*) on CROHME 2014 test set

Output label	Ground truth label (no. of nodes with this label)	No. of occurrences (percentage)
x	X (60)	46 (76.7% = 46/60)
x	$\times$ (145)	26 (17.93%)
p	P (30)	24 (80%)
,	1 (794)	19 (2.39%)
c	C (31)	16 (51.61%)
y	Y (26)	14 (53.85%)
$+$	t (162)	14 (8.64%)
.	… (123)	13 (10.57%)
X	x (890)	16 (1.80%)
a	x (890)	14 (1.57%)
1	| (62)	11 (17.74%)
-	1(794)	10 (1.26%)
$\times$	x (890)	10 (1.12%)
z	2 (713)	10 (1.40%)
q	9 (98)	10 (10.20%)

We only list the cases that occur at least 10 times

Another improvement would be to integrate explicitly a language model to promote frequent symbols.

Edge label Table 11 provides the edge (SLG) label errors of CROHME 2014 test set using (*Merge 9*). As can be seen, a large amount of errors come from the last row which represents the missing edges. 1858 edges with label *Right* are missed in our system, along with 929 segmentation edges. In addition, we can see the errors of high frequency in the sixth row which represents that five relationship (exclude *Right*) edges or segmentation edges or *NoRelation* (_) edges are mis-classified as *Right* edges. Among them, 1600 *NoRelation* (_) edges are recognized as *Right*. The post-process step of *Make connected SRT* is one of the reasons since we take a hard decision (add *Right* edges) in this step. Another possible reason is that, as *Right* relationship is the most frequent relation in math expressions, the classifiers may answer too often this frequent class.

We explore deeper the problem of the missing edges which appear in the last row of Table 11. In fact, there exist three sources which result in the missing edges: (1) the edges are missed at the graph representation stage. We evaluated the graph model in Sect. 5.2.1 where around 6% relationships were missed. One of the future works could be searching for a better graph representation model to catch the 6% missing relationships. (2) Some edges in the derived graph are recognized by the system as *NoRelation* (_), which actually have a ground truth label of one of 6 relationship or symbol (segmentation edge). (3) Even if we derive multiple trees from the graph G, they do not well cover the graph completely. Thus, a better strategy for deriving trees from the graph will be explored in future works.

We illustrate one test sample ($\frac{9}{9+\sqrt{9}}$) from CROHME 2014 test set recognized by system (*Merge9*). We provide the handwritten input, along with the built SLG for this test sample (Fig. 16). For this expression, the structure of it was correctly recognized, only one error being the first symbol '9' of the denominator was recognized as '$\rightarrow$'. This error belongs to the type of the similar-look error we have explained in error analysis section. Enlarging the training data set could be a solution to solve it. Also, it could be eased by introducing a language model since $\frac{9}{\rightarrow+\sqrt{9}}$ is not a valid expression from this point of view.

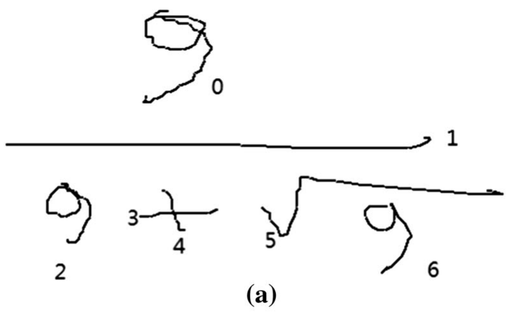
6 Conclusion

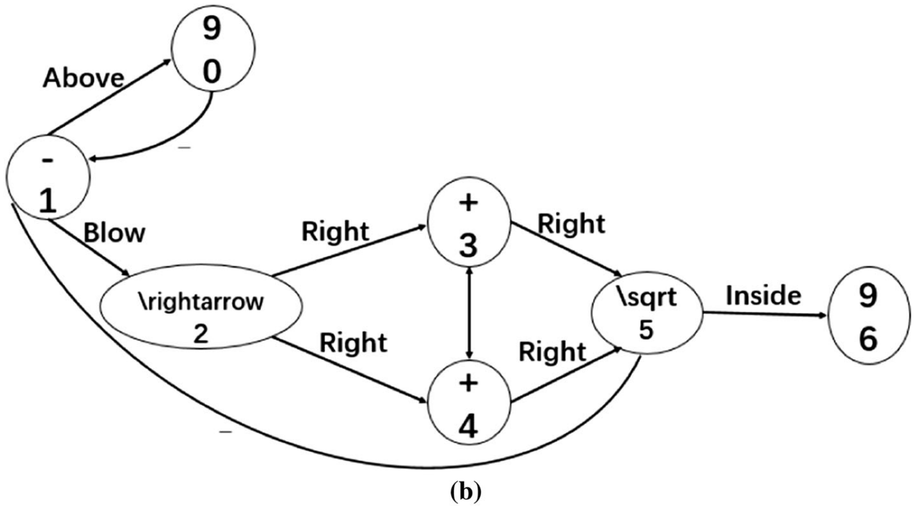
In this work, we developed a tree-BLSTM-based recognition system for online handwritten mathematical expression. To be able to process 2-D languages, we extend the

Table 11 Illustration of edge (SLG) label errors of (*Merge 9*) on CROHME 2014 test set

	* (9044)	Above (592)	Below (627)	Inside (377)	Right (13698)	Sub (1115)	Sup (923)	_ (261528)
*	208	0	0		17	1	1	29
Above	8		1				21	10
Below	2			1	1	114		7
Inside		5	1			1		9
Right	344	65	22	40		152	112	1600
Sub	4		6	3	44		1	7
Sup	1	3			35	3		31
_	929	300	80	109	1858	189	235	

The first column represents the output labels; the first row offers the ground truth labels, as well as the number of edges with each label; other cells in this table provide the corresponding no. of occurrences. '*' represents segmentation edges, grouping two nodes into a symbol. The label of segmentation edge is a symbol (For convenient representation, we do not give the specific symbol types, but an overall label '*')





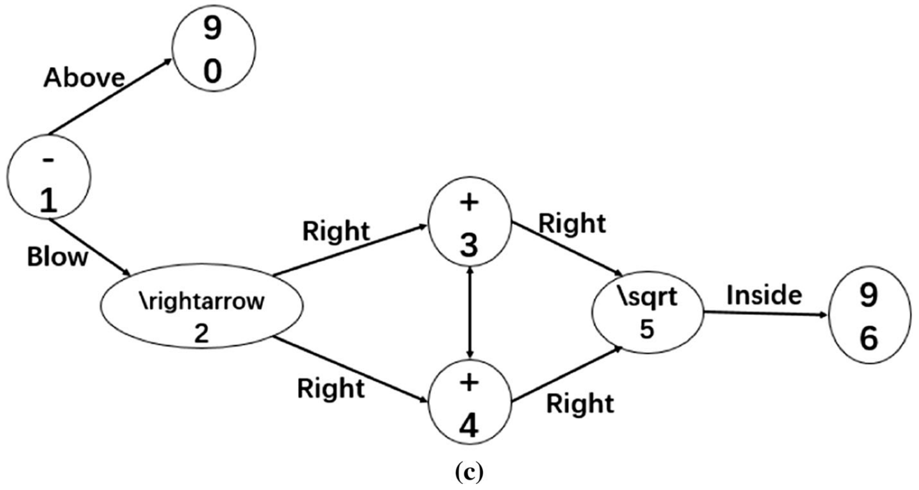


Fig. 16 **a** $\frac{9}{9+\sqrt{9}}$ written with 7 strokes, **b** the SLG after merging several trees and performing other post-processing steps, **c** the SLG with *NoRelation* edges removed. There is a node label error: the stroke 2 with the ground truth label '9' was wrongly classified as '$\rightarrow$'

chain-structured BLSTM to tree-based BLSTM which could model the dependencies in a tree structure, and extend CTC to local CTC to relatively constrain the outputs and at the same time benefit from introducing an additional 'blank' class. One major difference with the traditional approaches is that there is no explicit segmentation, recognition and layout extraction steps but a unique trainable system that produces directly a SLG describing a mathematical expression. The proposed system, without using any grammar, achieves competitive results in online math expression recognition domain.

Based on the current method and error analysis, we summarize several possible directions for future work. Some work should be done with regards to improve the existing method, like improving the graph model, proposing a better strategy for deriving trees and developing a stronger post-process stage. Some efforts could be put into introducing language model into the graph. For example, as known an n-gram model is widely used in 1-D language processing like text and speech, how to take into account the statistical properties of n-grams in math expression recognition task is an interesting direction to explore in future. Another interesting work could be to extend BLSTM model to a DAG structure which will better cover the derived graph and therefore be able to handle more contextual information compared to the tree structure BLSTM. The current recognition system achieves competitive results without using any grammar knowledge. In future, we could apply graph grammar to improve the current recognition rate. In this paper, we extend the chain-structured BLSTM to a tree topology to let it model the dependency directly in a tree structure. Furthermore, we extend the CTC training technique to local CTC to constrain the output position relatively at the same time improve the system training efficiency compared to frame-wise training. These proposed algorithms are generic ones and could be apply in other research fields such as diagram or flowchart recognition.

Acknowledgements We would like to thank the China Scholarship Council (CSC) for supporting PhD studentship at Université de Nantes.

Compliance with ethical standards

Conflict of interest All authors declare that no support, financial or otherwise, has been received from any organization that may have an interest in the submitted work and there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Kremer R (1988) Visual languages for knowledge representation. In: 11th workshop on knowledge acquisition, modeling and management
2. Marriott K, Meyer B, Wittenburg KB (1998) A survey of visual language specification and recognition. In: Marriott K, Meyer B (eds) Visual language theory. Springer, New York, pp 5–85
3. Zanibbi R, Blostein D (2012) Recognition and retrieval of mathematical expressions. Int J Doc Anal Recognit 15(4):331–357
4. Mouchère H, Zanibbi R, Garain U, Viard-Gaudin C (2016) Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. Int J Doc Anal Recognit 19(2):173–189
5. Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J (2009) A novel connectionist system for unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 31(5):855–868
6. Álvaro Muñoz F, Sánchez JA, Benedí JM (2013) Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In: 12th international conference on document analysis and recognition (ICDAR), pp 1012–1016
7. Zhang H, Li J, Ji Y, Yue H (2017) Understanding subtitles by character-level sequence-to-sequence learning. IEEE Trans Ind Inform 13(2):616–624
8. Tai KS, Socher R, Manning CD (2015) Improved semantic representations from tree-structured long short-term memory networks. arXiv:1503.00075
9. Zhu XD, Sobhani P, Guo HY (2015) Long short-term memory over recursive structures. In: International conference on machine learning (ICML), pp 1604–1612
10. Zhu XD, Sobhani P, Guo HY (2016) Dag-structured long short-term memory for semantic compositionality. In: Proceedings of NAACL-HLT, pp 917–926
11. Zhang T, Mouchère H, Viard-Gaudin C (2016) Online handwritten mathematical expressions recognition by merging multiple 1D interpretations. In: 15th international conference on frontiers in handwriting recognition (ICFHR), pp 187–192
12. Zhang T, Mouchère H, Viard-Gaudin C (2016) Using BLSTM for interpretation of 2-D languages. Doc Numér Lavoisier 19(2):135–157
13. Anderson RH (1967) Syntax-directed recognition of hand-printed two-dimensional mathematics. In: Symposium on interactive systems for experimental applied mathematics, pp 436–459
14. Chang SK (1970) A method for the structural analysis of two-dimensional mathematical expressions. Inf Sci 2(3):253–272
15. Martin WA (1971) Computer input/output of mathematical expressions. In: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, pp 78–89
16. Anderson RH (1977) Two-dimensional mathematical notation. In: Fu KS (ed) Syntactic pattern recognition, applications. Communication and cybernetics, vol 14. Springer, Berlin, Heidelberg
17. Blostein D, Grbavec A (1997) Recognition of mathematical notation. Handbook of character recognition and document image analysis 21:557–582
18. Chan KF, Yeung DY (2000) Mathematical expression recognition: a survey. Int J Doc Anal Recognit 3(1):3–15
19. Tapia E, Rojas R (2007) A survey on recognition of on-line handwritten mathematical notation. Tech report, Free University of Berlin, Department of Computer Science
20. Chou PA (1989) Recognition of equations using a two-dimensional stochastic context-free grammar. Vis Commun Image Process IV 1199:852–863
21. Koschinski M, Winkler HJ, Lang M (1995) Segmentation and recognition of symbols within handwritten mathematical expressions. In: International conference on acoustics, speech, and signal processing (ICASSP), pp 2439–2442
22. Winkler HJ, Fahrner H, Lang M (1995) A soft-decision approach for structural analysis of handwritten mathematical expressions. In: International conference on acoustics, speech, and signal processing (ICASSP), pp 2459–2462
23. Lehmberg S, Winkler HJ, Lang M (1996) A soft-decision approach for symbol segmentation within handwritten mathematical expressions. In: IEEE international conference on acoustics, speech, and signal processing (ICASSP), 3434–3437
24. Matsakis NE (1999) Recognition of handwritten mathematical expressions. PhD thesis, Massachusetts Institute of Technology
25. Zanibbi R, Blostein D, Cordy JR (2002) Recognizing mathematical expressions using tree transformation. IEEE Trans Pattern Anal Mach Intell 24(11):1455–1467
26. Tapia E, Rojas R (2003) Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: International workshop on graphics recognition, pp 329–340
27. Toyozumi K, Yamada N, Kitasaka T, Mori K, Suenaga Y, Mase K, Takahashi T (2004) A study of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information. In: 17th international conference on pattern recognition (ICPR), pp 630–633

28. Tapia E (2005) Understanding mathematics: a system for the recognition of on-line handwritten mathematical expressions. PhD thesis, Freie Universität Berlin
29. Zhang L, Blostein D, Zanibbi R (2005) Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In: Eighth international conference on document analysis and recognition, pp 972–976
30. Shi Y, Li HY, Soong FK (2007) A unified framework for symbol segmentation and recognition of handwritten mathematical expressions. In: Ninth International conference on document analysis and recognition (ICDAR), pp 854–858
31. Aly W, Uchida S, Fujiyoshi A, Suzuki M (2009) Statistical classification of spatial relationships among mathematical symbols. In: 10th international conference on document analysis and recognition (ICDAR), pp 1350–1354
32. Yamamoto R, Sako S, Nishimoto T, Sagayama S (2006) On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. In: Tenth international workshop on frontiers in handwriting recognition, 249–254
33. Awal AM, Mouchère H, Viard-Gaudin C (2014) A global learning approach for an online handwritten mathematical expression recognition system. Pattern Recognit Lett 35(1):68–77
34. Álvaro Muñoz F, Sánchez JA, Benedí JM (2014) Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. Pattern Recognit Lett 35(1):58–67
35. Álvaro Muñoz F, Sánchez JA, Benedí JM (2016) An integrated grammar-based approach for mathematical expression recognition. Pattern Recognit 51:135–147
36. MacLean S, Labahn G (2013) A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. Int J Doc Anal Recognit 16(2):139–163
37. Celik M, Yanikoglu B (2011) Probabilistic mathematical formula recognition using a 2d context-free graph grammar. In: International conference on document analysis and recognition (ICDAR), pp 161–166
38. Julca-Aguilar F (2016) Recognition of online handwritten mathematical expressions using contextual information. PhD thesis, Université de Nantes, Université Bretagne Loire, Universidade de São Paulo
39. Deng YT, Kanervisto A, Rush AM (2016) What you get is what you see: a visual markup decompiler. arXiv:1609.04938
40. Zhang JS, Du J, Zhang SL, Liu D, Hu YL, Hu JS, Wei S, Dai LR (2017) Watch, attend and parse: an end-to-end neural network based approach to handwritten mathematical expression recognition. Pattern Recognit 71:196–206
41. Zanibbi R, Mouchère H, Viard-Gaudin C (2013) Evaluating structural pattern recognition for handwritten math via primitive label graphs. In: Proc. SPIE 8658, document recognition and retrieval XX, p 865817
42. Graves A (2012) Supervised sequence labelling with recurrent neural networks. Springer, Berlin
43. Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J (2001) Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. IEEE Press, Washington
44. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
45. Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional LSTM networks. IEEE Int Joint Conf Neural Netw 2005:2047–2052
46. Graves A, Jaitly N, Mohamed AR (2013) Hybrid speech recognition with deep bidirectional LSTM. In: IEEE workshop on automatic speech recognition and understanding (ASRU), pp 273–278
47. Graves A, Schmidhuber J (2009) Offline handwriting recognition with multidimensional recurrent neural networks. Adv Neural Inf Process Syst, 545–552
48. Messina R, Louradour J (2015) Segmentation-free handwritten Chinese text recognition with LSTM-RNN. In: 13th International conference on document analysis and recognition (ICDAR), pp 171–175
49. Bluche T, Louradour J, Messina R (2016) Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention. arXiv:1604.03286
50. Maalej R, Kherallah M (2016) Improving MDLSTM for offline Arabic handwriting recognition using dropout at different positions. In: International conference on artificial neural networks, pp. 431–438
51. Maalej R, Tagougui N, Kherallah M (2016) Recognition of handwritten Arabic words with dropout applied in MDLSTM. In: International conference image analysis and recognition, pp 746–752
52. Álvaro Muñoz F (2015) Mathematical expression recognition based on probabilistic grammars. PhD thesis, Universitat Politècnica de València
53. Bluche T, Ney H, Louradour J, Kermorvant C (2015) Framewise and CTC training of Neural Networks for handwriting recognition. In: 13th International conference on document analysis and recognition (ICDAR), pp 81–85
54. Álvaro Muñoz F, Sánchez JA, Benedí JM (2014) Offline features for classifying handwritten math symbols with recurrent neural networks. In: 2014 22nd international conference on pattern recognition (ICPR), pp 2944–2949
55. Mouchère H, Viard-Gaudin C, Zanibbi R, Garain U (2014) Crohme 2014: competition on recognition of on-line handwritten mathematical expressions. In: 2014 14th international conference on frontiers in handwriting recognition (ICFHR), pp 791–796
56. Hu L (2016) Features and algorithms for visual parsing of handwritten mathematical expressions. PhD thesis, Rochester Institute of Technology