UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Ting ZHANG

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique
Spécialité : Informatique et applications
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le 26 Octobre 2017

New Architectures for Handwritten Mathematical Expressions Recognition

Acknowledgments

Thanks to the various encounters and choices in life, I could have an experience studying in France at a fairly young age. Along the way, I met a lot of beautiful people and things.

Christian and Harold, you are so nice professors. This thesis would not have been possible without your considerate guidance, advice and encouragement. Thank you for sharing your knowledge and experience, for reading my papers and thesis over and over and providing meaningful comments. Your serious attitude towards work has a deep impact on me, today and tomorrow. Harold, thanks for your help in technique during the 3 years' study. Thank all the colleagues from IVC/IRCCyN or IPI/LS2N for giving me such a nice working environment, for so many warm moments, for giving me help when I need some one to speak French to negotiate on the phone, many times. Suiyi and Zhaoxin, thanks for being rice friends with me each lunch in Polytech. Thanks all the friends I met in Nantes for so much laughing, so many colorful weekends with you. Also, I would like to thank the China Scholarship Council (CSC) for supporting 3 years' PhD studentship at Université de Nantes. Finally, thank my parents, little brother and my grandparents for their understanding, support to my study, and endless love to me.

In addition, I would like to thank the members of the dissertation committee for accepting being either examiner or reviewer, and putting efforts on reviewing this thesis.

Contents

List of Tables **7**

List of Figures **9**

List of Abbreviations **13**

1 Introduction **15**
- 1.1 Motivation . . . 15
- 1.2 Mathematical expression recognition . . . 16
- 1.3 The proposed solution . . . 19
- 1.4 Thesis structure . . . 23

I State of the art **25**

2 Mathematical expression representation and recognition **27**
- 2.1 Mathematical expression representation . . . 27
  - 2.1.1 Symbol level: Symbol relation (layout) tree . . . 27
  - 2.1.2 Stroke level: Stroke label graph . . . 30
  - 2.1.3 Performance evaluation with stroke label graph . . . 31
- 2.2 Mathematical expression recognition . . . 33
  - 2.2.1 Overall review . . . 34
  - 2.2.2 The recent integrated solutions . . . 35
  - 2.2.3 End-to-end neural network based solutions . . . 39
  - 2.2.4 Discussion . . . 39

3 Sequence labeling with recurrent neural networks **43**
- 3.1 Sequence labeling . . . 43
- 3.2 Recurrent neural networks . . . 44
  - 3.2.1 Topology . . . 44
  - 3.2.2 Forward pass . . . 46
  - 3.2.3 Backward pass . . . 46
  - 3.2.4 Bidirectional networks . . . 47
- 3.3 Long short-term memory (LSTM) . . . 48
  - 3.3.1 Topology . . . 48
  - 3.3.2 Forward pass . . . 49
  - 3.3.3 Backward pass . . . 50
  - 3.3.4 Variants . . . 51
- 3.4 Connectionist temporal classification (CTC) . . . 53
  - 3.4.1 From outputs to labelings . . . 53
  - 3.4.2 Forward-backward algorithm . . . 53

3.4.3 Loss function 55
3.4.4 Decoding 56

II Contributions 59

4 Mathematical expression recognition with single path 61
4.1 From single path to stroke label graph 62
4.1.1 Complexity of expressions 62
4.1.2 The proposed idea 62
4.2 Detailed Implementation 64
4.2.1 BLSTM Inputs 65
4.2.2 Features 65
4.2.3 Training process — local connectionist temporal classification 66
4.2.4 Recognition Strategies 69
4.3 Experiments 70
4.3.1 Data sets 71
4.3.2 Experiment 1: theoretical evaluation 71
4.3.3 Experiment 2 73
4.3.4 Experiment 3 76
4.4 Discussion 76

5 Mathematical expression recognition by merging multiple paths 79
5.1 Overview of graph representation 79
5.2 The framework 82
5.3 Detailed implementation 83
5.3.1 Derivation of an intermediate graph G 83
5.3.2 Graph evaluation 84
5.3.3 Select paths from G 86
5.3.4 Training process 87
5.3.5 Recognition 87
5.3.6 Merge paths 87
5.4 Experiments 89
5.5 Discussion 94

6 Mathematical expression recognition by merging multiple trees 95
6.1 Overview: Non-chain-structured LSTM 95
6.2 The proposed Tree-based BLSTM 96
6.3 The framework 99
6.4 Tree-based BLSTM for online mathematical expression recognition 99
6.4.1 Derivation of an intermediate graph G 99
6.4.2 Graph evaluation 103
6.4.3 Derivation of trees from G 103
6.4.4 Feed the inputs of the Tree-based BLSTM 105
6.4.5 Training process 106
6.4.6 Recognition process 110
6.4.7 Post process 110
6.5 Experiments 111
6.5.1 Experiment 1 111
6.5.2 Experiment 2 112
6.5.3 Experiment 3 115

6.6 Error analysis 117
6.7 Discussion 118

7 Conclusion and future works 125
7.1 Conclusion 125
7.2 Future works 127

Bibliography 129

Publications 135

List of Tables

2.1 Illustration of the terminology related to recall and precision. 33

4.1 The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels on the *time* path). 73
4.2 The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels on the *time* path). 73
4.3 The symbol level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results. 74
4.4 The expression level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results. 74
4.5 The symbol level evaluation results (mean values) on CROHME 2014 test set with different training and decoding methods, features. 76
4.6 The standard derivations of the symbol level evaluation results on CROHME 2014 test set with local CTC training and maximum decoding method, 5 local features. 76

5.1 The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels of the nodes and edges of the built graph). 86
5.2 The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels of the nodes and edges of the built graph). 86
5.3 Illustration of the used classifiers in the different experiments depending of the type of path. 89
5.4 The symbol level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results. 90
5.5 The expression level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results 91

6.1 The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels). 103
6.2 The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels). 103
6.3 The different types of the derived trees. 105
6.4 The symbol level evaluation results on CROHME 2014 test set with *Tree-Time* only. . . . 112
6.5 The expression level evaluation results on CROHME 2014 test set with *Tree-Time* only. . 112
6.6 The symbol level evaluation results on CROHME 2014 test set with 3 trees, along with CROHME 2014 participant results. 113
6.7 The expression level evaluation results on CROHME 2014 test set with 3 trees, along with CROHME 2014 participant results. 114
6.8 The symbol level evaluation results on CROHME 2016 test set with the system of *Merge 9*, along with CROHME 2016 participant results. 115
6.9 The expression level evaluation results on CROHME 2016 test set with the system of *Merge 9*, along with CROHME 2016 participant results. 115
6.10 The symbol level evaluation results on CROHME 2014 test set with 11 trees. 116

6.11 The expression level evaluation results on CROHME 2014 test set with 11 trees. 116
6.12 Illustration of node (SLG) label errors of (*Merge 9*) on CROHME 2014 test set. We only list the cases that occur ≥ 10 times. 117
6.13 Illustration of edge (SLG) label errors of (*Merge 9*) on CROHME 2014 test set. 118

List of Figures

1.1 Illustration of mathematical expression examples. (a) A simple and liner expression consisting of only left-right relationship. (b) A 2-D expression where left-right, above-below, superscript relationships are involved. 16
1.2 Illustration of expression $z^d + z$ written with 5 strokes. 17
1.3 Illustration of the symbol segmentation of expression $z^d + z$ written with 5 strokes. 17
1.4 Illustration of the symbol recognition of expression $z^d + z$ written with 5 strokes. 18
1.5 Illustration of the structural analysis of expression $z^d + z$ written with 5 strokes. $Sup : Superscript, R : Right$. 18
1.6 Illustration of the symbol relation tree of expression $z^d + z$. $Sup : Superscript, R : Right$. 18
1.7 Illustration of the structural analysis of expression $z^d + z$ written with 5 strokes connected by 4 pen up strokes. 19
1.8 Illustration of the proposal of recognizing ME expressions with a single path. 20
1.9 Illustration of the proposal of recognizing ME expressions by merging multiple paths. . . . 21
1.10 Illustration of the proposal of recognizing ME expressions by merging multiple trees. . . . 22

2.1 Symbol relation tree (a) and operator tree (b) of expression $(a+b)^2$. $Sup : Superscript, R : Right, Arg : Argument$. 28
2.2 The symbol relation tree (SRT) for (a) $\frac{a+b}{c}$, (b) $a + \frac{b}{c}$. 'R' refers to $Right$ relationship. . . 28
2.3 The symbol relation trees (SRT) for (a) $\sqrt[3]{x}$, (b) $\sum_{i=0}^{n} x^i$ and (c) $\int_x xdx$. 'R' refers to $Right$ relationship while 'Sup' and 'Sub' denote $Superscript$ and $Subscript$ respectively. 29
2.4 Math file encoding for expression $(a + b)^2$. (a) Presentation MathML; (b) LaTeX. Adapted from [Zanibbi and Blostein, 2012]. 29
2.5 (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 30
2.6 The file formats for representing SLG considering the expression in Figure2.5a. (a) The file format taking stroke as the basic entity. (b) The file format taking symbol as the basic entity. 31
2.7 Adjacency Matrices for Stroke Label Graph. (a) The adjacency matrix format: li denotes the label of stroke si and eij is the label of the edge from stroke si to stroke sj. (b) The adjacency matrix of labels corresponding to the SLG in Figure 2.5c. 32
2.8 '$2+2$' written with four strokes was recognized as '$2-1^2$'. (a) The SLG of the recognition result; (b) the corresponding adjacency matrix. 'Sup' denotes $Superscript$ relationship. . . 32
2.9 Example of a search for most likely expression candidate using the CYK algorithm. Extracted from [Yamamoto et al., 2006]. 35
2.10 The system architecture proposed in [Awal et al., 2014]. Extracted from [Awal et al., 2014]. 36
2.11 A simple example of Fuzzy r-CFG. Extracted from [MacLean and Labahn, 2013]. 37
2.12 (a) An input handwritten expression; (b) a shared parse forest of (a) considering the grammar depicted in Figure 2.11. Extracted from [MacLean and Labahn, 2013] 38
2.13 Geometric features for classifying the spatial relationship between regions B and C. Extracted from [Álvaro et al., 2016] 39

2.14 Achitecture of the recognition system proposed in [Julca-Aguilar, 2016]. Extracted from [Julca-Aguilar, 2016] . . . 40
2.15 Network architecture of *WYGIWYS*. Extracted from [Deng et al., 2016] . . . 41

3.1 Illustration of sequence labeling task with the examples of handwriting (top) and speech (bottom) recognition. Input signals is shown on the left side while the ground truth is on the right. Extracted from [Graves et al., 2012]. . . . 44
3.2 A multilayer perceptron. . . . 45
3.3 A recurrent neural network. The recurrent connections are highlighted with red color. . . . 45
3.4 An unfolded recurrent network. . . . 45
3.5 An unfolded bidirectional network. Extracted from [Graves et al., 2012]. . . . 48
3.6 LSTM memory block with one cell. Extracted from [Graves et al., 2012]. . . . 49
3.7 A deep bidirectional LSTM network with two hidden levels. . . . 52
3.8 (a) A chain-structured LSTM network; (b) A tree-structured LSTM network with arbitrary branching factor. Extracted from [Tai et al., 2015]. . . . 52
3.9 Illustration of CTC forward algorithm. Blanks are represented with black circles and labels are white circles. Arrows indicate allowed transitions. Adapted from [Graves et al., 2012]. 54
3.10 Mistake incurred by best path decoding. Extracted from [Graves et al., 2012]. . . . 57
3.11 Prefix search decoding on the alphabet {X, Y}. Extracted from [Graves et al., 2012]. . . . 57

4.1 Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME. . . . 61
4.2 Illustration of the complexity of math expressions. . . . 62
4.3 (a) The $time$ path (red) in SLG; (b) the SLG obtained by using the $time$ path; (c) the post-processed SLG of '2 + 2', added edges are depicted as bold. . . . 63
4.4 (a) P^{eo} written with four strokes; (b) the SRT of P^{eo}; (c) r^2h written with three strokes; (d) the SRT of r^2h, the red edge cannot be generated by the time sequence of strokes . . . 64
4.5 The illustration of on-paper points (blue) and in-air points (red) in $time$ path, a_1+a_2 written with 6 strokes. . . . 65
4.6 The illustration of (a) θ_i, ϕ_i and (b) ψ_i used in feature description. The points related to feature computation at p_i are depicted in red. . . . 66
4.7 The possible sequences of point labels in one stroke. . . . 67
4.8 Local CTC forward-backward algorithm. Black circles represent labels and white circles represent blanks. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated in the reverse direction. . . . 68
4.9 Illustration for the decision of the label of strokes. As stroke 5 and 7 have the same label, the label of stroke 6 could be '$+$', '_' or one of the six relationships. All the other strokes are provided with the ground truth labels in this example. . . . 70
4.10 Real examples from CROHME 2014 data set. (a) sample from **Data set 1**; (b) sample from **Data set 2**; (c) sample from **Data set 3**. . . . 72
4.11 (a) $a \geq b$ written with four strokes; (b) the built SLG of $a \geq b$ according to the recognition result, all labels are correct. . . . 75
4.12 (a) $44 - \frac{4}{4}$ written with six strokes; (b) the ground-truth SLG; (c) the rebuilt SLG according to the recognition result. Three edge errors occurred: the $Right$ relation between stroke 2 and 4 was missed because there is no edge from stroke 2 to 4 in the time path; the edge from stroke 4 to 3 was missed for the same reason; the edge from stroke 2 to 3 was wrongly recognized and it should be labeled as $NoRelation$. . . . 75

5.1 Examples of graph models. (a) An example of minimum spanning tree at stroke level. Extracted from [Matsakis, 1999]. (b) An example of Delaunay-triangulation-based graph at symbol level. Extracted from [Hirata and Honda, 2011]. . . . 80
5.2 An example of line of sight graph for a math expression. Extracted from [Hu, 2016]. . . . 81

5.3 Stroke representation. (a) The bounding box. (b) The convex hull. 81
5.4 Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME. 82
5.5 Illustration of visibility between a pair of strokes. s_1 and s_3 are visible to each other. . . . 83
5.6 Five directions for a stroke s_i. Point (0, 0) is the center of bounding box of s_i. The angle of each region is $\frac{\pi}{4}$. 84
5.7 (a)$\frac{d}{dx}a^x$ is written with 8 strokes; (b) the SLG built from raw input using the proposed method; (c) the SLG from ground truth; (d) illustration of the difference between the built graph and the ground truth graph, red edges denote the unnecessary edges and blue edges refer to the missed ones compared to the ground truth. 85
5.8 Illustration of the strategy for merge. 88
5.9 (a) $a \geq b$ written with four strokes; (b) the derived graph from the raw input; (c) the labeled graph (provided the label and the related probability) with merging 7 paths; (d) the built SLG after post process, all labels are correct. 92
5.10 (a) $44 - \frac{4}{4}$ written with six strokes; (b) the derived graph; (c) the built SLG by merging several paths; (d) the built SLG with $NoRelation$ edges removed. 93

6.1 (a) A chain-structured LSTM network; (b) A tree-structured LSTM network with arbitrary branching factor. Extracted from [Tai et al., 2015]. 96
6.2 A tree based structure for chains (from root to leaves). 96
6.3 A tree based structure for chains (from leaves to root). 97
6.4 Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME. 99
6.5 Illustration of visibility between a pair of strokes. s_1 and s_3 are visible to each other. . . . 100
6.6 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 $[-\frac{\pi}{8}, \frac{\pi}{8}]$; $R2 : [\frac{\pi}{8}, \frac{3*\pi}{8}]$; $R3 : [\frac{3*\pi}{8}, \frac{7*\pi}{8}]$; $R4 : [-\frac{7*\pi}{8}, -\frac{3*\pi}{8}]$; $R5 : [-\frac{3*\pi}{8}, -\frac{\pi}{8}]$. 100
6.7 (a) $\frac{f}{a} = \frac{b}{f}$ is written with 10 strokes; (b) create nodes; (c) add $Crossing$ edges. $C : Crossing$. 101
6.7 (d) add $R1, R2, R3, R4, R5$ edges; (e) add $Time$ edges. $C : Crossing, T : Time$. 102
6.8 (a) $\frac{f}{a} = \frac{b}{f}$ is written with 10 strokes; (b) the derived graph G, the red part is one of the possible trees with $s2$ as the root. $C : Crossing, T : Time$. 104
6.9 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. 105
6.10 A tree-based BLSTM network with one hidden level. We only draw the full connection on one short sequence (red) for a clear view. 106
6.11 Illustration for the pre-computation stage of tree-based BLSTM. (a) From the input layer to the hidden layer (from root to leaves), (b) from the input layer to the hidden layer (from leaves to root). 108
6.12 The possible labels of points in one short sequence. 108
6.13 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. This figure is a local part (limited in one stroke) of Figure 4.8. 109
6.14 Possible relationship conflicts existing in merging results. 111
6.15 (a) $a \geq b$ written with four strokes; (b) the derived graph; (b) *Tree-Time*; (c)*Tree-Left-R1* (In this case, *Tree-0-R1* is the same as *Tree-Left-R1*); (e) the built SLG of $a \geq b$ after merging several trees and performing other post process steps, all labels are correct; (f) the built SLG with $NoRelation$ edges removed. 119
6.16 (a) $44 - \frac{4}{4}$ written with six strokes; (b) the derived graph; (b) *Tree-Time*; (c)*Tree-Left-R1* (In this case, *Tree-0-R1* is the same as *Tree-Left-R1*); 120
6.16 (b)the built SLG after merging several trees and performing other post process steps; (c) the built SLG with $NoRelation$ edges removed. 121

6.17 (a) $\frac{9}{9+\sqrt{9}}$ written with 7 strokes; (b) the derived graph; (b) *Tree-Time*; 122
6.17 (d)*Tree-Left-R1*; (e)*Tree-0-R1*; (f)the built SLG after merging several trees and performing other post process steps; (g) the built SLG with $NoRelation$ edges removed. There is a node label error: the stroke 2 with the ground truth label '9' was wrongly classified as '$\rightarrow$'. 123

List of Abbreviations

2D-PCFGs	Two-Dimensional Probabilistic Context-Free Grammars.
AC	Averaged Center.
ANNs	Artificial Neural Networks.
BAR	Block Angle Range.
BB	Bounding Box.
BBC	Bounding Box Center.
BLSTM	Bidirectional Long Short-Term Memory.
BP	Back Propagation.
BPTT	Back Propagation Through Time.
BRNNs	Bidirectional Recurrent Neural Networks (BRNNs).
CH	Convex Hull.
CNN	Convolutional Neural Network.
CPP	Closest Point Pair.
CROHME	Competition on Recognition of Handwritten Mathematical Expressions.
CTC	Connectionist Temporal Classification.
CYK	Cock Younger Kasami.
DT	Delaunay Triangulation.
FNNs	Feed-forward Neural Networks.
HMM	Hidden Markov Model.
KNN	K Nearest Neighbor.
LOS	Line Of Sight.
ME	Mathematical Expression.
MLP	Multilayer Perceptron.
MST	Minimum Spanning Tree.
r-CFG	Relational Context-Free Grammar.
RNN	Recurrent Neural Network.
RTRL	Real Time Recurrent Learning.
SLG	Stroke Label Graph.

SRT Symbol Relation Tree.

TS Time Series.

UAR Unblocked Angle Range.

VAR Visibility Angle Range.

1

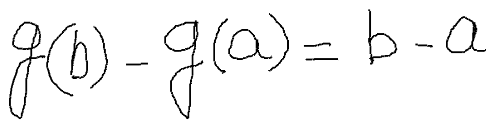
Introduction

In this thesis, we explore the idea of online handwritten Mathematical Expression (ME) interpretation using Bidirectional Long Short-Term Memory (BLSTM) and Connectionist Temporal Classification (CTC) topology, and finally build a graph-driven recognition system, bypassing the high time complexity and manual work with the classical grammar-driven systems. Advanced recurrent neural network BLSTM with a CTC output layer achieved great success in sequence labeling tasks, such as text and speech recognition. However, the move from sequence recognition to mathematical expression recognition is far from being straightforward. Unlike text or speech where only left-right (or past-future) relationship is involved, ME has a 2 dimensional (2-D) structure consisting of relationships like subscript and superscript. To solve this recognition problem, we propose a graph-driven system, extending the chain-structured BLSTM to a tree structure topology allowing to handle the 2-D structure of ME, and extending CTC to local CTC to relatively constrain the outputs.

In the first section of the this chapter, we introduce the motivation of our work from both the research point and the practical application point. Section 1.2 provides a global view of the mathematical expression recognition problem, covering some basic concepts and the challenges involved in it. Then in Section 1.3, we describe the proposed solution concisely, to offer the readers an overall view of main contributions of this work. The thesis structure will be presented in the end of the chapter.

1.1 Motivation

A visual language is defined as any form of communication that relies on two- or three-dimensional graphics rather than simply (relatively) linear text [Kremer, 1998]. Mathematical expressions, plans and musical notations are commonly used cases in visual languages [Marriott et al., 1998]. As an intuitive and easily (relatively) comprehensible knowledge representation model, mathematical expression (Figure 1.1) could help the dissemination of knowledge in some related domains and therefore is 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 the 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 preparation of scientific document. Thus, there comes the problem of handwritten mathematical expression recognition. Incidentally, the recent large developments of touch screen devices also drive the research of this field.


(a)

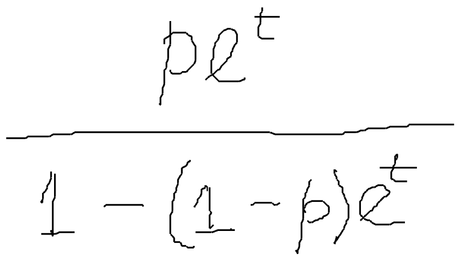

(b)

Figure 1.1 – Illustration of mathematical expression examples. (a) A simple and liner expression consisting of only left-right relationship. (b) A 2-D expression where left-right, above-below, superscript relationships are involved.

Handwritten mathematical expression recognition is an appealing topic in pattern recognition field since it exhibits a big research challenge and underpins many practical applications. From a scientific point of view, a large set of symbols (more than 100) needs to be recognized, and also the 2 dimensional (2-D) structures (specifically the relationships between a pair of symbols, for example $superscript$ and $subscript$), both of which increase the difficulty of this recognition problem. With regard to the application, it offers an easy and direct way to input MEs into computers, and therefore improves productivity for scientific writers.

Research on the recognition of math notation began in the 1960's [Anderson, 1967], and several research publications are available in the following thirty years [Chang, 1970, Martin, 1971, Anderson, 1977]. Since the 90's, with the large developments of touch screen devices, this field has started to be active, gaining amounts of research achievement and considerable attention from the research community. A number of surveys [Blostein and Grbavec, 1997, Chan and Yeung, 2000, Tapia and Rojas, 2007, Zanibbi and Blostein, 2012] summarize the proposed techniques for math notation recognition. This research domain has been boosted by the Competition on Recognition of Handwritten Mathematical Expressions (CROHME) [Mouchère et al., 2016], 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. It attracts increasing participation of research groups from all over the world. In this thesis, the provided data and evaluation tools from CROHME will be used and results will be compared to participants.

1.2 Mathematical expression recognition

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 thesis will be focused on online handwritten ME recognition.

For the online case, a handwritten mathematical expression could have one or more strokes and a stroke is a sequence of points sampled from the trajectory of the writing tool between a pen-down and a pen-up at a fixed interval of time. For example, the expression $z^d + z$ shown in Figure 1.2 is written with 5 strokes, two strokes of which belong to the symbol '$+$'.

Generally, ME recognition involves three tasks [Zanibbi and Blostein, 2012]:
(1) **Symbol Segmentation**, which consists in grouping strokes that belong to the same symbol. In Figure 1.3, we illustrate the segmentation of the expression $z^d + z$ where $stroke3$ and $stroke4$ are grouped as a

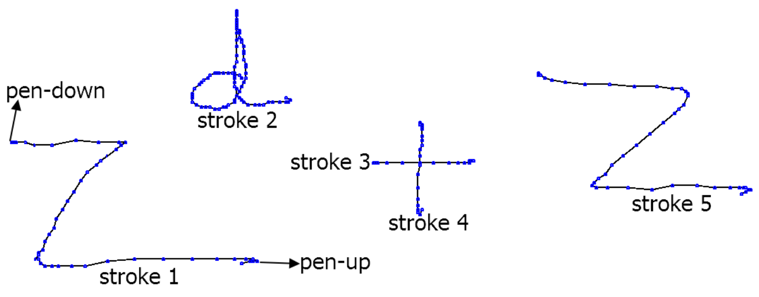


Figure 1.2 – Illustration of expression $z^d + z$ written with 5 strokes.

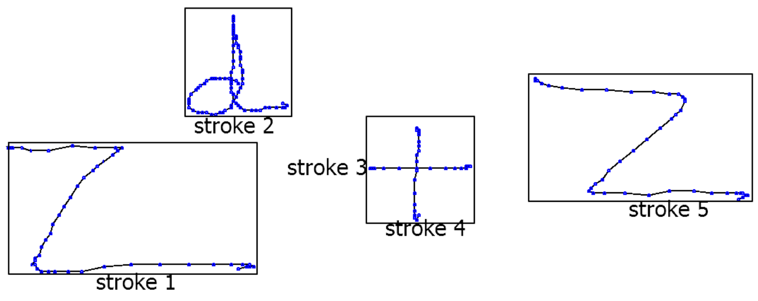


Figure 1.3 – Illustration of the symbol segmentation of expression $z^d + z$ written with 5 strokes.

symbol candidate. This task becomes very difficult in the presence of delayed strokes, which occurs when interspersed symbols are written. For example, it could be possible in the real case that someone write first a part of the symbol '$+$' ($stroke3$), and then the symbol 'z' ($stroke5$), in the end complete the other part of the symbol '$+$' ($stroke4$). Thus, in fact any combination of any number of strokes could form a symbol candidate. It is exhausting to take into account each possible combination of strokes, especially for complex expressions having a large number of strokes.
(2) **Symbol Recognition**, the task of labeling the symbol candidates to assign each of them a symbol class. Still considering the same sample $z^d + z$, Figure 1.4 presents the symbol recognition of it. This is as well a difficult task because the number of classes is quite important, more than one hundred different symbols including digits, alphabet, operators, Greek letters and some special math symbols; it exists an overlapping between some symbol classes: (1) for instance, digit '0', Greek letter 'θ', and character 'O' might look about the same when considering different handwritten samples (inter-class variability); (2) there is a large intra-class variability because each writer has his own writing style. Being an example of inter-class variability, the $stroke5$ in Figure 1.4 looks like and could be recognized as 'z', 'Z' or '2'. To address these issues, it is important to design robust and efficient classifiers as well as a large training data set. Nowadays, most of the proposed solutions are based on machine learning algorithms such as neural networks or support vector machines.
(3) **Structural Analysis**, its goal is to identify spatial relations between symbols and with the help of a 2-D language to produce a mathematical interpretation, such as a symbol relation tree which will be emphasized in later chapter. For instance, the $Superscript$ relationship between the first 'z' and 'd', and the $Right$ relationship between the first 'z' and '$+$' as illustrated in Figure 1.5. Figure 1.6 provides the corresponding symbol relation tree which is one of the possible ways to represent math expressions. Structural analysis strongly depends on the correct understanding of relative positions among symbols. Most approaches consider only local information (such as relative symbol positions and their sizes) to determine the relation between a pair of symbols. Although some approaches have proposed the use of contextual information to improve system performances, modeling and using such information is still challenging.

These three tasks can be solved sequentially or jointly. In the early stages of the study, most of the proposed solutions [Chou, 1989, Koschinski et al., 1995, Winkler et al., 1995, Matsakis, 1999, Zanibbi et al., 2002, Tapia and Rojas, 2003, Tapia, 2005, Zhang et al., 2005] are sequential ones which treat the

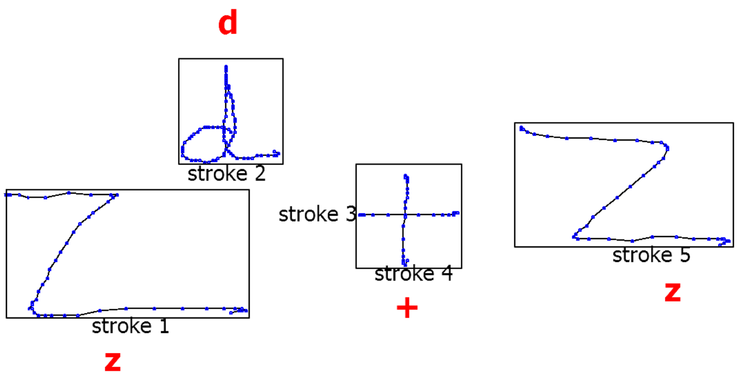


Figure 1.4 – Illustration of the symbol recognition of expression $z^d + z$ written with 5 strokes.

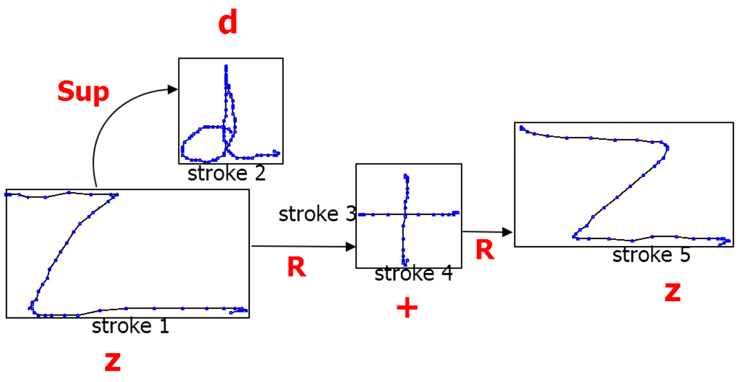


Figure 1.5 – Illustration of the structural analysis of expression $z^d + z$ written with 5 strokes. $Sup : Superscript, R : Right.$

$$\begin{tikzcd}
 & d & \\
z \arrow[ru, "Sup"] \arrow[r, "R"] & + \arrow[r, "R"] & z
\end{tikzcd}$$

Figure 1.6 – Illustration of the symbol relation tree of expression $z^d + z$. $Sup : Superscript, R : Right.$

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. 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 global structure, and vice versa.

The recent proposed solutions, considering the natural relationship between the three tasks, perform 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 [Yamamoto et al., 2006, Awal et al., 2014, Álvaro et al., 2014b, 2016, MacLean and Labahn, 2013] and graph grammar [Celik and Yanikoglu, 2011, Julca-Aguilar, 2016]) parsing techniques, producing expressions conforming to the rules of a manually defined grammar. Either string or graph grammar parsing, each one has a high computational complexity.

In conclusion, generally the current state of the art systems are grammar-driven solutions. For these grammar-driven solutions, it requires not only a large amount of manual work for defining grammars, but also a high computational complexity for grammar parsing process. As an alternative approach, we propose to explore a non grammar-driven solution for recognizing math expression. This is the main goal of this thesis, 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.

1.3 The proposed solution

As well known, Bidirectional Long Short-term Memory (BLSTM) network with a Connectionist Temporal Classification (CTC) output layer achieved great success in sequence labeling tasks, such as text and speech recognition. This success is due to the LSTM's ability of capturing long-term dependency in a sequence and the effectiveness of CTC training method. Unlike the grammar-driven solutions, the new architectures proposed in this thesis include contextual information with BLSTM instead of grammar parsing technique. In this thesis, we will explore the idea of using the sequence-structured BLSTM with a CTC stage to recognize 2-D handwritten mathematical expression.

Mathematical expression recognition with a single path. As a first step to try, we consider linking the last point and the first point of a pair of strokes successive in the input time to allow the handwritten ME to be handled with BLSTM topology. As shown in Figure 1.7, after processing, the original 5 visible strokes

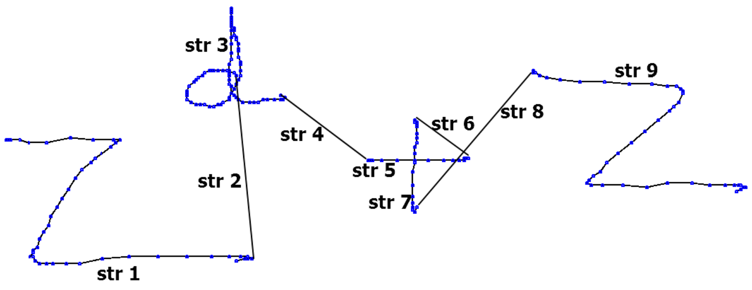


Figure 1.7 – Illustration of the structural analysis of expression $z^d + z$ written with 5 strokes connected by 4 pen up strokes.

turn out to be 9 strokes; in fact, they could be regarded as a global sequence, just as same as the regular 1-D text. We would like to use these later added strokes to represent the relationships between pairs of stokes by assigning them a ground truth label. The remaining work is to train a model using this global sequence with a BLSTM and CTC topology, and then label each stroke in the global sequence. Finally, with the sequence of outputted labels, we explore how to build a 2-D expression. The framework is illustrated in Figure 1.8.

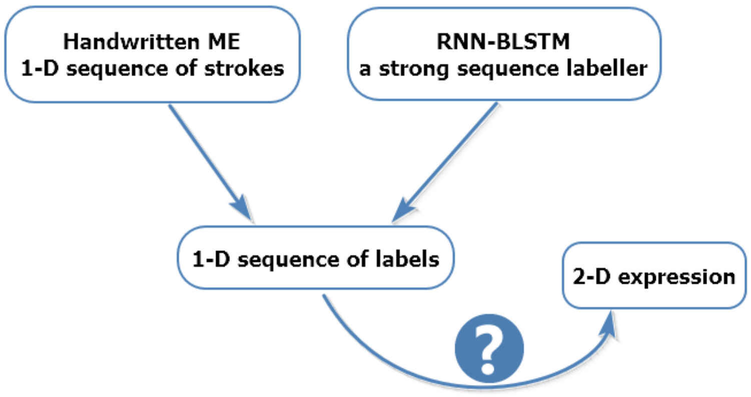


Figure 1.8 – Illustration of the proposal of recognizing ME expressions with a single path.

Mathematical expression recognition by merging multiple paths. Obviously, the solution of linking only pairs of strokes successive in the input time could handle just some relatively simple expressions. For complex expressions, some relationships could be missed such as the $Right$ relationship between $stroke1$ and $stroke5$ in Figure 1.7. Thus, we turn to a graph structure to model the relationships between strokes in mathematical expressions. We illustrate this new proposal in Figure 1.9. As shown, the input of the recognition system is an handwritten expression which is a sequence of strokes; the output is the stroke label graph which consists of the information about the label of each stroke and the relationships between stroke pairs. As the first step, we derive an intermediate graph from the raw input considering both the temporal and spatial information. In this graph, each node is a stroke and edges are added according to temporal or spatial properties between strokes. We assume that strokes which are close to each other in time and space have a high probability to be a symbol candidate. Secondly, several 1-D paths will be selected from the graph since the classifier model we are considering is a sequence labeller. Indeed, a classical BLSTM-RNN model is able to deal with only sequential structure data. Next, we use the BLSTM classifier to label the selected 1-D paths. This stage consists of two steps —— the training and recognition process. Finally, we merge these labeled paths to build a complete stroke label graph.

Mathematical expression recognition by merging multiple trees. Human beings interpret handwritten math expression considering the global contextual information. However, in the current system, even though several paths from one expression are taken into account, each of them is considered individually. The classical BLSTM model could access information from past and future in a long range but the information outside the single sequence is of course not accessible to it. Thus, we would like to develop a neural network model which could handle directly a structure not limited to a chain. With this new neural network model, we could take into account the information in a tree instead of a single path at one time when dealing with one expression.

We extend the chain-structured BLSTM to tree structure topology and apply this new network model for online math expression recognition. Figure 1.10 provides a global view of the recognition system. Similar to the framework presented in Figure 1.9, we first drive an intermediate graph from the raw input. Then, instead of 1-D paths, we consider from the graph deriving trees which will be labeled by tree-based BLSTM model as a next step. In the end, these labeled trees will be merged to build a stroke label graph.

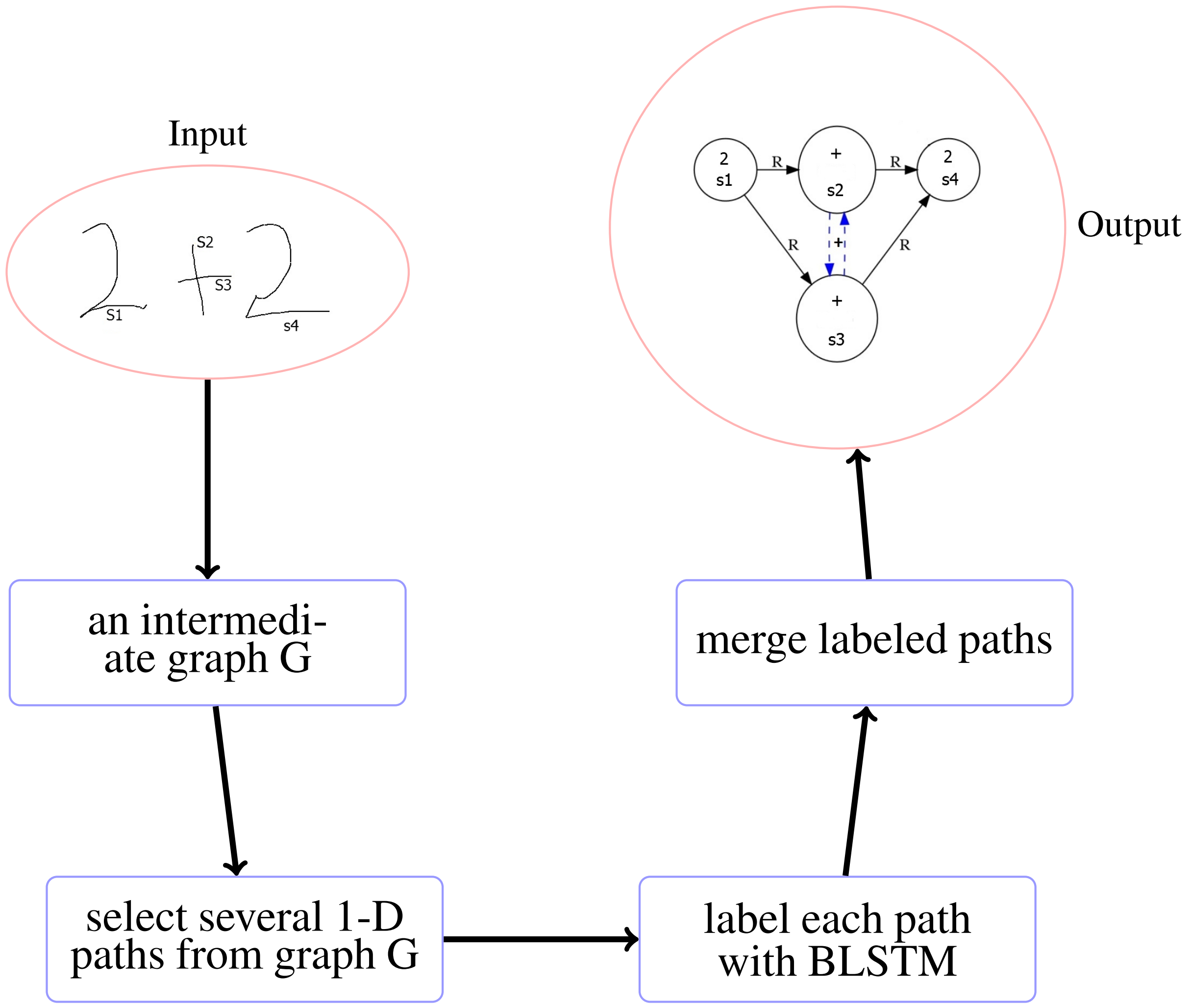


Figure 1.9 – Illustration of the proposal of recognizing ME expressions by merging multiple paths.

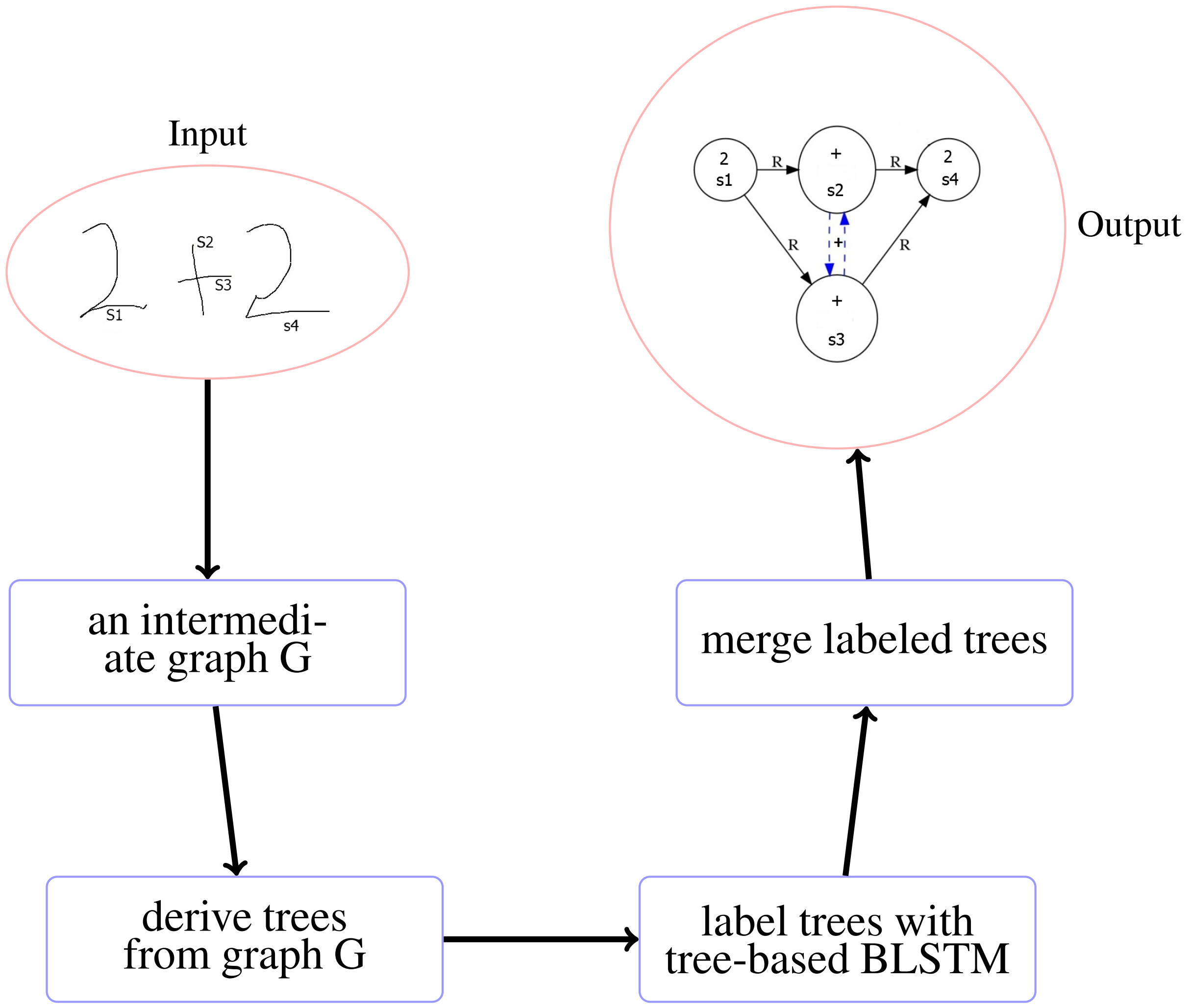


Figure 1.10 – Illustration of the proposal of recognizing ME expressions by merging multiple trees.

1.4 Thesis structure

Chapter 2 describes the previous works on ME representation and recognition. With regards to representation, we introduce the symbol relation tree (symbol level) and the stroke label graph (stroke level). Furthermore, as an extension, we describe the performance evaluation based on stroke label graph. For ME recognition, we first review the entire history of this research subject, and then only focus on more recent solutions which are used for a comparison with the new architectures proposed in this thesis.

Chapter 3 is focused on sequence labeling using recurrent neural networks, which is the foundation of our work. First of all, we explain the concept of sequence labeling and the goal of this task shortly. Then, the next section introduces the classical structure of recurrent neural network. The property of this network is that it can memorize contextual information but the range of the information could be accessed is quite limited. Subsequently, long short-term memory is presented with the aim of overcoming the disadvantage of the classical recurrent neural network. The new architecture is provided with the ability of accessing information over long periods of time. Finally, we introduce how to apply recurrent neural network for the task of sequence labeling, including the existing problems and the solution to solve them, i.e. the connectionist temporal classification technology.

In Chapter 4, we explore the idea of recognizing ME expressions with a single path. Firstly, we globally introduce the proposal that builds stroke label graph from a sequence of labels, along with the existing limitations in this stage. Then, the entire process of generating the sequence of labels with BLSTM and local CTC given the input is presented in detail, including firstly feeding the inputs of BLSTM, then the training and recognition stages. Finally, the experiments and discussion are described. One main drawback of the strategy proposed in this chapter is that only stroke combinations in time series are used in the representation model. Thus, some relationships are missed at the modeling stage.

In Chapter 5, we explore the idea of recognizing ME expressions by merging multiple paths, as a new model to overcome some limitations in the system of Chapter 4. The proposed solution will take into account more possible stroke combinations in both time and space such that less relationships will be missed at the modeling stage. We first provide an overview of graph representation related to build a graph from raw mathematical expression. Then we globally describe the framework of mathematical expression recognition by merging multiple paths. Next, all the steps of the recognition system are explained one by one in detail. Finally, the experiment part and the discussion part are presented respectively. One main limitation is that we use the classical chain-structured BLSTM to label a graph-structured input data.

In Chapter 6, we explore the idea of recognizing ME expressions by merging multiple trees, as a new model to overcome the limitation of the system of Chapter 5. We extend the chain-structured BLSTM to tree structure topology and apply this new network model for online math expression recognition. Firstly, a short overview with regards to the non-chain-structured LSTM is provided. Then, we present the new proposed neural network model named tree-based BLSTM. Next, the framework of ME recognition system based on tree-based BLSTM is globally introduced. Hereafter, we focus on the specific techniques involved in this system. Finally, experiments and discussion parts are covered respectively.

In Chapter 7, we conclude the main contributions of this thesis and give some thoughts about future work.

I

State of the art

2

Mathematical expression representation and recognition

This chapter introduces the previous works regarding to ME representation and ME recognition. In the first part, we will review the different representation models on symbol and stroke level respectively. On symbol level, symbol relation (layout) tree is the one we mainly focus on; on stroke level, we will introduce stroke label graph which is a derivation of symbol relation tree. Note that stroke label graph is the final output form of our recognition system. As an extension, we also describe the performance evaluation based on stroke label graph. In the second part, we review first the history of this recognition problem, and then put emphasize on more recent solutions which are used for a comparison with the new architectures proposed in this thesis.

2.1 Mathematical expression representation

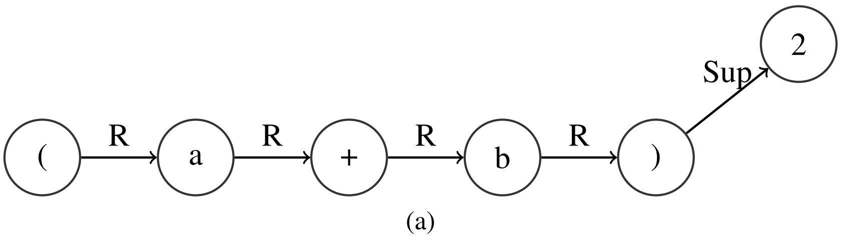
Structures can be depicted at three different levels: symbolic, object and primitive [Zanibbi et al., 2013]. In the case of handwritten ME, the corresponding levels are expression, symbol and stroke.

In this section, we will first introduce two representation models of math expression at the symbol level, especially Symbol Relation Tree (SRT). From the SRT, if going down to the stroke level, a Stroke Label Graph (SLG) could be derived, which is the current official model to represent the ground-truth of handwritten math expressions and also for the recognition outputs in Competitions CROHME.

2.1.1 Symbol level: Symbol relation (layout) tree

It is possible to describe a ME at the symbol level using a layout-based SRT, as well as an operator tree which is based on operator syntax. Symbol layout tree represents the placement of symbols on baselines (writing lines), and the spatial arrangement of the baselines [Zanibbi and Blostein, 2012]. As shown in Figure 2.1a, symbols '(', 'a', '+', 'b', ')' share a writing line while '2' belongs to the other writing line. An operator tree represents the operator and relation syntax for an expression [Zanibbi and Blostein, 2012]. The operator tree for $(a + b)^2$ shown in Figure 2.1b represents the addition of 'a' and 'b', squared. We will focus only on the model of symbol relation tree in the coming content since it is closely related to our work.

In SRT, nodes represent symbols, while labels on the edges indicate the relationships between symbols. For example, in Figure 2.2a, 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 Figure 2.2b, the symbol 'a' is the root; the symbol '+' is on the



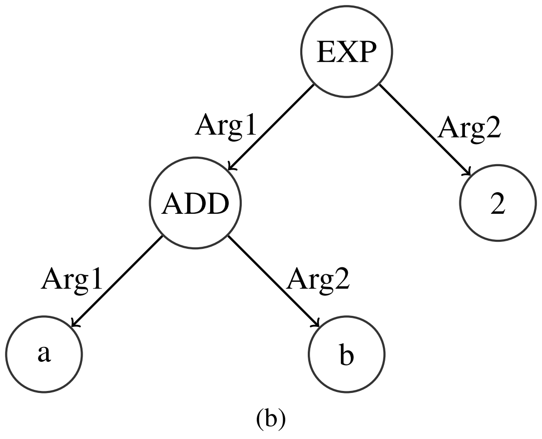


Figure 2.1 – Symbol relation tree (a) and operator tree (b) of expression $(a + b)^2$. $Sup : Superscript, R : Right, Arg : Argument.$

$Right$ of 'a'. As a matter of fact, the node inherits the spatial relationships of its ancestor. In Figure 2.2a, node '+' inherits the $Above$ relationship of its ancestor 'a'. Thus, '+' is also $Above$ '-' as 'a'. Similarly, 'b' is on the $Right$ of 'a' and $Above$ the '-'. Note that all the inherited relationships are ignored when we depict the SRTs in this work. This will be also the case in the evaluation stage since knowing the original edges is enough to ensure a proper representation.

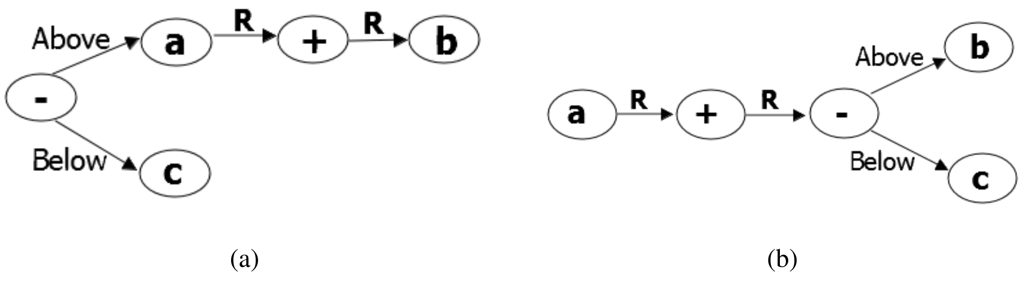


Figure 2.2 – The symbol relation tree (SRT) for (a) $\frac{a+b}{c}$, (b) $a + \frac{b}{c}$. 'R' refers to $Right$ relationship.

101 classes of symbols have been collected 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), $Superscript$, $Subscript$. For the case of nth-Roots, like $\sqrt[3]{x}$ as illustrated in Figure 2.3a, we define that the symbol '3' is $Above$ the square root and 'x' is $Inside$ the square root. The limits of an integral and summation are designated as $Above$ or $Superscript$ and $Below$ or $Subscript$ depending on the actual position of the bounds. For example, in expression $\sum\limits_{i=0}^{n} a^i$, 'n' is $Above$ the '$\sum$' and 'i' is $Below$ the '$\sum$' (Figure 2.3b). When we consider another case $\sum_{i=0}^{n} a^i$, 'n' is $Superscript$ the '$\sum$' and 'i' is $Subscript$ the '$\sum$'. The same strategy is held for the limits of integral. As can be seen in Figure 2.3c, the first 'x' is $Subscript$ the '$\int$' in the expression $\int_x xdx$.

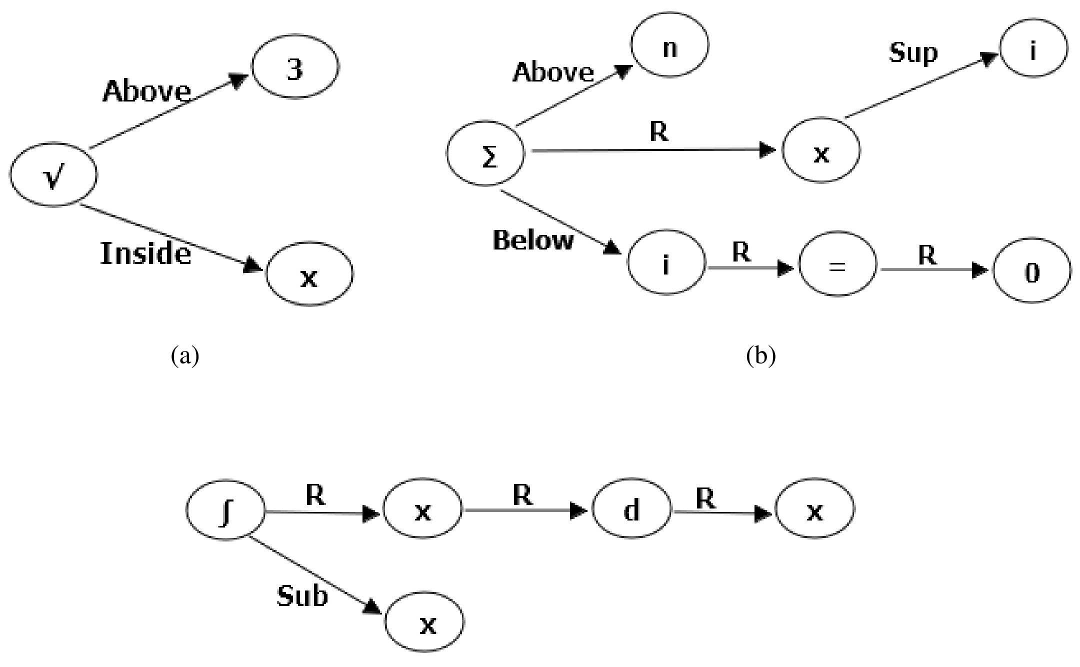


Figure 2.3 – The symbol relation trees (SRT) for (a) $\sqrt[3]{x}$, (b) $\sum_{i=0}^{n} x^i$ and (c) $\int_x xdx$. 'R' refers to $Right$ relationship while 'Sup' and 'Sub' denote $Superscript$ and $Subscript$ respectively.

File formats for representing SRT

File formats for representing SRT include Presentation MathML[1] and LaTeX, as shown in Figure 2.4. Compared to LaTeX, Presentation MathML contains additional tags to identify symbols types; these are primarily for formatting [Zanibbi and Blostein, 2012]. By the way, there are several files encoding for operator trees, including Content MathML and OpenMath [Davenport and Kohlhase, 2009, Dewar, 2000].

```
<msup>
    <mfenced>
            <mi>a</mi>
            <mo>+</mo>
            <mi>b</mi>
    </mfenced>
    <mn>2</mn>
</msup>
```

(a)

```
(a+b)^2
```

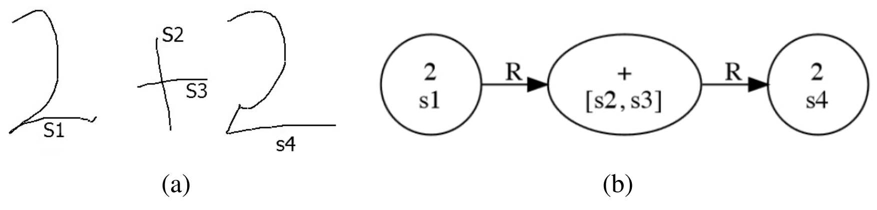
(b)

Figure 2.4 – Math file encoding for expression $(a+b)^2$. (a) Presentation MathML; (b) LaTeX. Adapted from [Zanibbi and Blostein, 2012].

1. Mathematical markup language (MathML) version 3.0, https://www.w3.org/Math/.

2.1.2 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 Figure 2.5a. The corresponding SRT and SLG are shown in Figure 2.5b and Figure 2.5c respectively. As Figure 2.5c illustrates, nodes of SLG are labeled with the class of the corresponding symbol to which the stroke belongs. A dashed edge



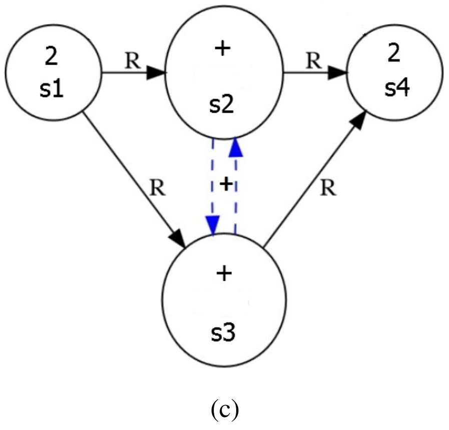


Figure 2.5 – (a) $2 + 2$ written with four strokes; (b) the symbol relation tree of $2 + 2$; (c) the SLG of $2 + 2$. The four strokes are indicated as s1, s2, s3, s4 in writing order. 'R' is for left-right relationship

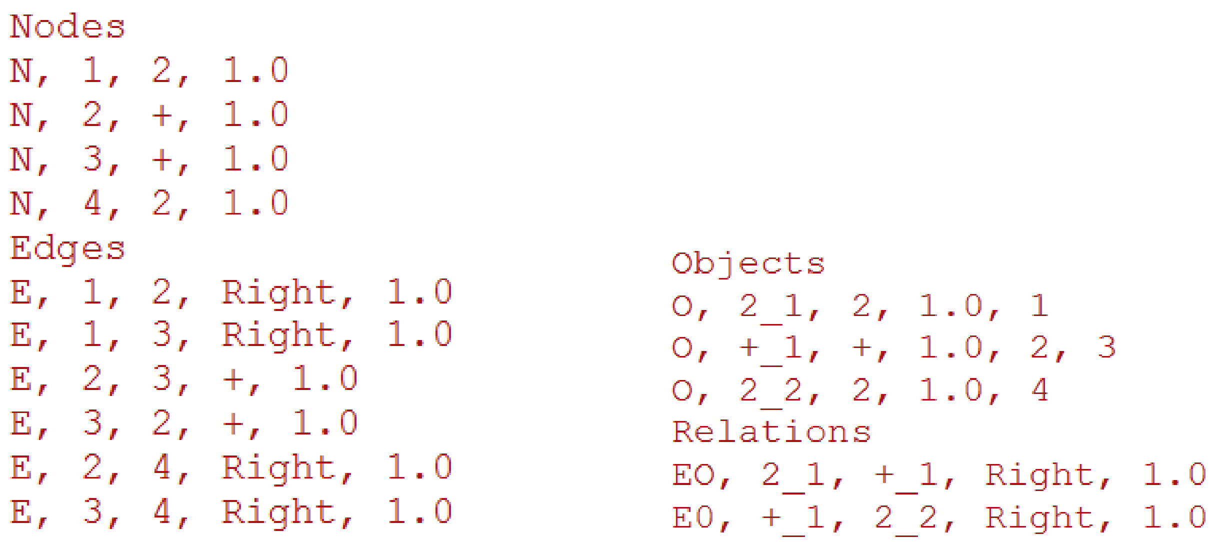
corresponds to segmentation information; it indicates that a pair of strokes belongs to the same symbol. In this case, the edge label is the same as the common symbol label. On the other hand, the non-dashed edges define spatial relationships between nodes and are labeled with one of the different possible 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 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 [Mouchère et al., 2016]. As the official format to represent the ground-truth of handwritten math expressions and also for the recognition outputs, it allows detailed error analysis on stroke, symbol and expression levels. In order to be comparable to the ground truth SLG and allow error analysis on any level, our recognition system aims to generate SLG from the input. It means that we need a label decision for each stroke and each stroke pair used in a symbol relation.

File formats for representing SLG

The file format we are using for representing SLG is illustrated with the example $2 + 2$ in Figure 2.6a. For each node, the format is like '$N, NodeIndex, NodeLabel, Probability$' where $Probability$ is always 1 in ground truth and depends on the classifier in system output. When it comes to edges, the format will be '$E, FromNodeIndex, ToNodeIndex, EdgeLabel, Probability$'.

An alternative format could be like the one shown in Figure 2.6b, which contains the same information as the previous one but with a more compact appearance. We take symbol as an individual to represent in this compact version but include the stroke level information also. For each object (or symbol), the format is '$O, ObjectIndex, ObjectLabel, Probability, StrokeList$' in which $StrokeList$' lists the indexes of the strokes this symbol consists of. Similarly, the representation for relationships is formatted as '$EO, FromObjectIndex, ToObjectIndex, RelationshipLabel, Probability$'.



```
Nodes
N, 1, 2, 1.0
N, 2, +, 1.0
N, 3, +, 1.0
N, 4, 2, 1.0
Edges
E, 1, 2, Right, 1.0
E, 1, 3, Right, 1.0
E, 2, 3, +, 1.0
E, 3, 2, +, 1.0
E, 2, 4, Right, 1.0
E, 3, 4, Right, 1.0
```

(a)

```
Objects
O, 2_1, 2, 1.0, 1
O, +_1, +, 1.0, 2, 3
O, 2_2, 2, 1.0, 4
Relations
EO, 2_1, +_1, Right, 1.0
EO, +_1, 2_2, Right, 1.0
```

(b)

Figure 2.6 – The file formats for representing SLG considering the expression in Figure2.5a. (a) The file format taking stroke as the basic entity. (b) The file format taking symbol as the basic entity.

2.1.3 Performance evaluation with stroke label graph

As mentioned in last section, both the ground truth and the recognition output of expression in CROHME are represented as SLGs. Then the problem of performance evaluation of a recognition system is essentially measuring the difference between two SLGs. This section will introduce how to compute the distance between two SLGs.

A SLG is a directed graph that can be visualized as an adjacency matrix of labels (Figure 2.7). Figure 2.7a provides the format of the adjacency matrix: the diagonal refers stroke (node) labels and other cells interpret stroke pair (edge) labels [Zanibbi et al., 2013]. Figure 2.7b presents the adjacency matrix of labels corresponding to the SLG in Figure 2.5c. The underscore '_' identifies that this edge exists and the label of it is $NoRelation$, or this edge does not exist. The edge $e14$ with the label of R is an inherited relationship which is not reflected in SLG as we said before. Suppose we have 'n' strokes in one expression, the number of cells in the adjacency matrix is n^2. Among these cells, 'n' cells represent the labels of strokes while the other '$n(n-1)$' cells interpret the segmentation information and relationships.

In order to analyze recognition errors in detail, Zanibbi et al. defined for SLGs a set of metrics in [Zanibbi et al., 2013]. They are listed as follows:

- ΔC, the number of stroke labels that differ.
- ΔS, the number of segmentation errors.
- ΔR, the number of spatial relationship errors.

	s1	s2	s3	s4
s1	**l1**	e12	e13	e14
s2	e21	**l2**	e23	e24
s3	e31	e32	**l3**	e34
s4	e41	e42	e43	**l4**

(a)

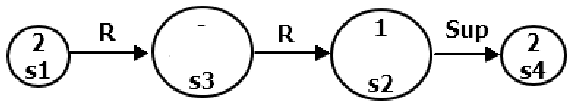
2	R	R	R
_	+	+	R
_	+	+	R
_	_	_	2

(b)

Figure 2.7 – Adjacency Matrices for Stroke Label Graph. (a) The adjacency matrix format: li denotes the label of stroke si and eij is the label of the edge from stroke si to stroke sj. (b) The adjacency matrix of labels corresponding to the SLG in Figure 2.5c.

- $\Delta L = \Delta S + \Delta R$, the number of edge labels that differ.
- $\Delta B = \Delta C + \Delta L = \Delta C + \Delta S + \Delta R$, the Hamming distance between the adjacency matrices.

Suppose that the sample '$2 + 2$' was interpreted as '$2 - 1^2$' as shown in Figure 2.8, we now compare the two adjacency matrices (the ground truth in Figure 2.7b and the recognition result in Figure 2.8b):



(a)

2	R	R	R
_	1	_	Sup
_	R	-	R
_	_	_	2

(b)

Figure 2.8 – '$2 + 2$' written with four strokes was recognized as '$2 - 1^2$'. (a) The SLG of the recognition result; (b) the corresponding adjacency matrix. 'Sup' denotes $Superscript$ relationship.

- $\Delta C = 2$, cells $l2$ and $l3$. The stroke $s2$ was wrongly recognized as 1 while $s3$ was incorrectly labeled as $-$.
- $\Delta S = 2$, cells $e23$ and $e32$. The symbol '$+$' written with 2 strokes was recognized as two isolated symbols.
- $\Delta R = 1$, cell $e24$. The $Right$ relationship was recognized as $Superscript$.
- $\Delta L = \Delta S + \Delta R = 2 + 1 = 3$.
- $\Delta B = \Delta C + \Delta L = \Delta C + \Delta S + \Delta R = 2 + 2 + 1 = 5$.

Zanibbi et al. defined two additional metrics at the expression level:

- $\Delta B_n = \frac{\Delta B}{n^2}$, the percentage of correct labels in adjacency matrix where 'n' is the number of strokes. ΔB_n is the Hamming distance normalized by the label graph size n^2.

- ΔE, the error averaged over three types of errors: $\Delta C, \Delta S, \Delta L$. As ΔS is part of ΔL, segmentation errors are emphasized more than other edge errors ΔR in this metric [Zanibbi et al., 2013].

$$\Delta E = \frac{\frac{\Delta C}{n} + \sqrt{\frac{\Delta S}{n(n-1)}} + \sqrt{\frac{\Delta L}{n(n-1)}}}{3} \tag{2.1}$$

We still consider the sample shown in Figure 2.8b, thus:

- $\Delta B_n = \frac{\Delta B}{n^2} = \frac{5}{4^2} = \frac{5}{16} = 0.3125$
-

$$\Delta E = \frac{\frac{\Delta C}{n} + \sqrt{\frac{\Delta S}{n(n-1)}} + \sqrt{\frac{\Delta L}{n(n-1)}}}{3} = \frac{\frac{2}{4} + \sqrt{\frac{2}{4(4-1)}} + \sqrt{\frac{3}{4(4-1)}}}{3} = 0.4694 \tag{2.2}$$

Given the representation form of SLG and the defined metrics, 'precision' and 'recall' rates at any level (stroke, symbol and expression) could be computed [Zanibbi et al., 2013], which are current indexes for accessing the performance of the systems in CROHME. 'recall' and 'precision' rates are commonly used to evaluate results in machine learning experiments [Powers, 2011]. In different research fields like information retrieval and classification tasks, different terminology are used to define 'recall' and 'precision'. However, the basic theory behind remains the same. In the context of this work, we use the case of segmentation results to explain 'recall' and 'precision' rates. To well define them, several related terms are given first as shown in Tabel 2.1. 'segmented' and 'not segmented' refer to the prediction of classifier while

Table 2.1 – Illustration of the terminology related to recall and precision.

	relevant	non relevant
segmented	true positive (tp)	false positive (fp)
not segmented	false negative (fn)	true negative (tn)

'relevant' and 'non relevant' refer to the ground truth. 'recall' is defined as

$$recall = \frac{tp}{tp + fn} \tag{2.3}$$

and 'precision' is defined as

$$precision = \frac{tp}{tp + fp} \tag{2.4}$$

In Figure 2.8, '$2+2$' written with four strokes was recognized as '$2-1^2$'. Obviously in this case, tp is equal to 2 since two '2' symbols were segmented and they exist in the ground truth. fp is equal to 2 also because '-' and '1' were segmented but they are not the ground truth. fn is equal to 1 as '+' was not segmented but it is the ground truth. Thus, 'recall' is $\frac{2}{2+1}$ and 'precision' is $\frac{2}{2+2}$. A larger 'recall' than 'precision' means the symbols are over segmented in our context.

2.2 Mathematical expression recognition

In this section, we first review the entire history of this research subject, and then only focus on more recent solutions which are provided as a comparison to the new architectures proposed in this thesis.

2.2.1 Overall review

Research on the recognition of math notation began in the 1960's [Anderson, 1967], and several research publications are available in the following thirty years [Chang, 1970, Martin, 1971, Anderson, 1977]. Since the 90's, with the large developments of touch screen devices, this field has started to be active, gaining amounts of research achievement and considerable attention from the research community. A number of surveys [Blostein and Grbavec, 1997, Chan and Yeung, 2000, Tapia and Rojas, 2007, Zanibbi and Blostein, 2012, Mouchère et al., 2016] summarize the proposed techniques for math notation recognition.

As described already in Section 1.2, ME recognition involves three interdependent tasks [Zanibbi and Blostein, 2012]: (1) Symbol segmentation, which consists in grouping strokes that belong to the same symbol; (2) symbol recognition, the task of labeling the symbol to assign each of them a symbol class; (3) structural analysis, its goal is to identify spatial relations between symbols and with the help of a grammar to produce a mathematical interpretation. These three tasks can be solved sequentially or jointly.

Sequential solutions. In the early stages of the study, most of the proposed solutions [Chou, 1989, Koschinski et al., 1995, Winkler et al., 1995, Lehmberg et al., 1996, Matsakis, 1999, Zanibbi et al., 2002, Tapia and Rojas, 2003, Toyozumi et al., 2004, Tapia, 2005, Zhang et al., 2005, Yu et al., 2007] 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 [Matsakis, 1999], Bayesian framework [Yu et al., 2007], graph-based method [Lehmberg et al., 1996, Toyozumi et al., 2004] and so on. The symbol classifiers used consist of Nearest Neighbor, Hidden Markov Model, Multi-layer Perceptron, Support Vector Machine, Recurrent neural networks and so on. For spatial relationship classification, the proposed features include symbol bounding box [Anderson, 1967], relative size and position [Aly et al., 2009], 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 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 [Yamamoto et al., 2006, Awal et al., 2014, Álvaro et al., 2014b, 2016, MacLean and Labahn, 2013] and graph grammar [Celik and Yanikoglu, 2011, Julca-Aguilar, 2016]) parsing techniques, producing expressions conforming to the rules of a manually defined grammar. String grammar parsing, along with graph grammar parsing, has a high time complexity in fact. In the next section we will analysis deeper these approaches. Instead of using grammar parsing technique, the new architectures proposed in this thesis 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 [Deng et al., 2016, Zhang et al., 2017]. 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 such as LaTeX. However, considering we are given trace information in the online case, despite the final LaTeX string, it is necessary to decide a label for each stroke. This information is not available now in end-to-end systems.

2.2.2 The recent integrated solutions

In [Yamamoto et al., 2006], a framework based on stroke-based stochastic context-free grammar is proposed for on-line handwritten mathematical expression recognition. They model handwritten mathematical expressions with a stochastic context-free grammar and formulate the recognition problem as a search problem of the most likely mathematical expression candidate, which can be solved using the Cock Younger Kasami (CYK) algorithm. With regard to the handwritten expression grammar, the authors define production rules for structural relation between symbols and also for a composition of two sets of strokes to form a symbol. Figure 2.9 illustrates the process of searching the most likely expression candidate with

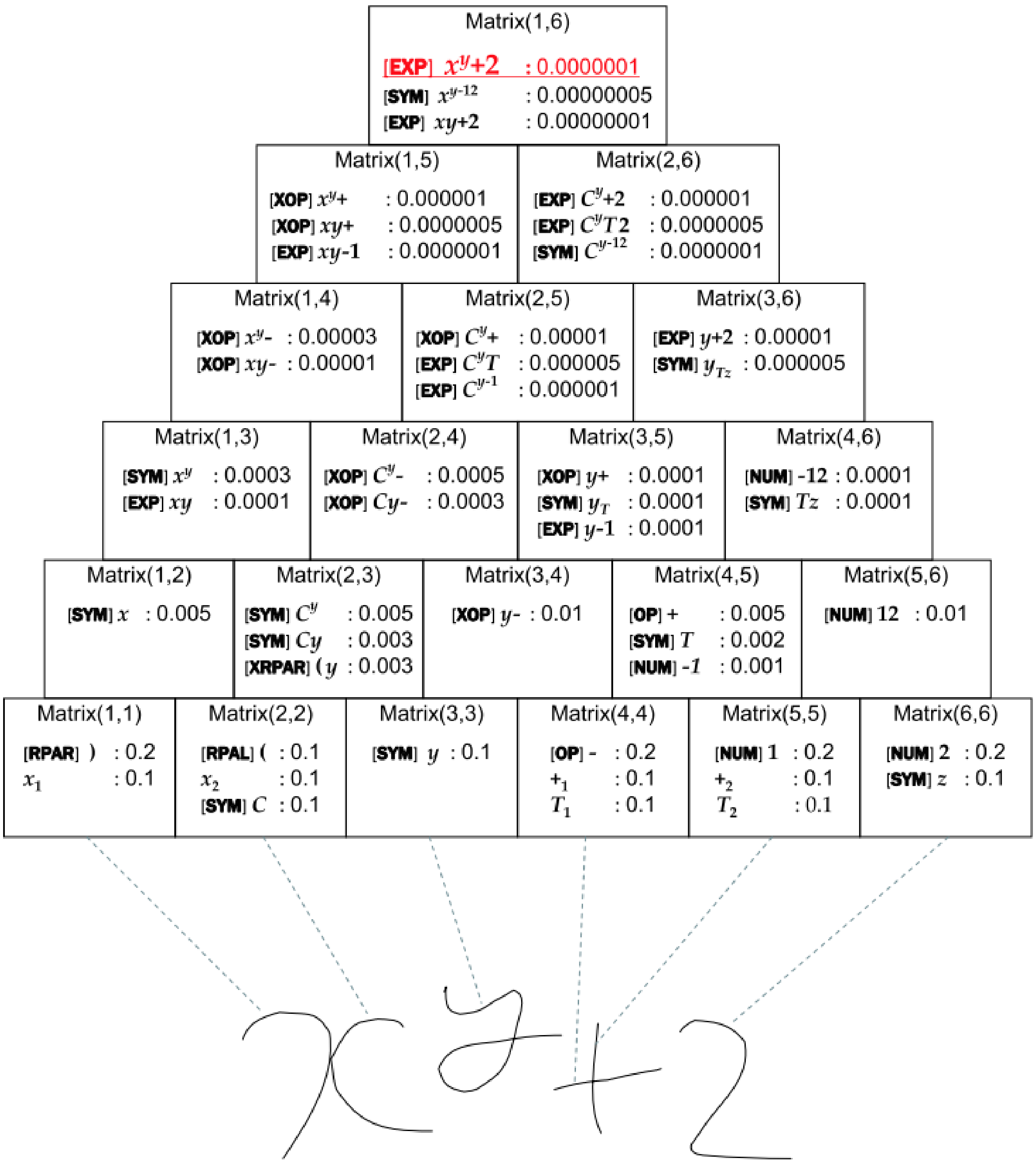


Figure 2.9 – Example of a search for most likely expression candidate using the CYK algorithm. Extracted from [Yamamoto et al., 2006].

the CYK algorithm on an example of $x^y + 2$. The algorithm which fill the CYK table from bottom to up is as following:

- For each input stroke i, corresponding to cell $Matrix(i, i)$ shown in Figure 2.9, the probability of each stroke label candidate is computed. This calculation is the same as the likelihood calculation in isolated character recognition. In this example, the 2 best candidates for the first stroke of the presented example are ')' with the probability of 0.2 and the first stroke of x (denoted as x_1 here) with the probability of 0.1.
- In cell $Matrix(i, i+1)$, the candidates for strokes i and $i+1$ are listed. As shown in cell $Matrix(1, 2)$ of the same example, the candidate x with the likelihood of 0.005 is generated with the production rule $< x \rightarrow x_1 x_2, SameSymbol >$. The structure likelihood computed using the bounding boxes is 0.5 here. Then the product of stroke and structure likelihoods is $0.1 \times 0.1 \times 0.5 = 0.005$.
- Similarly, in cell $Matrix(i, i + k)$, the candidates for strokes from i to $i + k$ are listed with the corresponding likelihoods.
- Finally, the most likely EXP candidate in cell $Matrix(1, n)$ is the recognition result.

In this work, they assume that symbols are composed only of consecutive (in time) strokes. In fact, this assumption does not work with the cases when the delayed strokes take place.

In [Awal et al., 2014], the recognition system handles mathematical expression recognition as a simultaneous optimization of expression segmentation, symbol recognition, and 2D structure recognition under the restriction of a mathematical expression grammar. The proposed approach is a global strategy allowing learning mathematical symbols and spatial relations directly from complete expressions. The general architecture of the system in illustrated in Figure 2.10. First, a symbol hypothesis generator based on 2-D

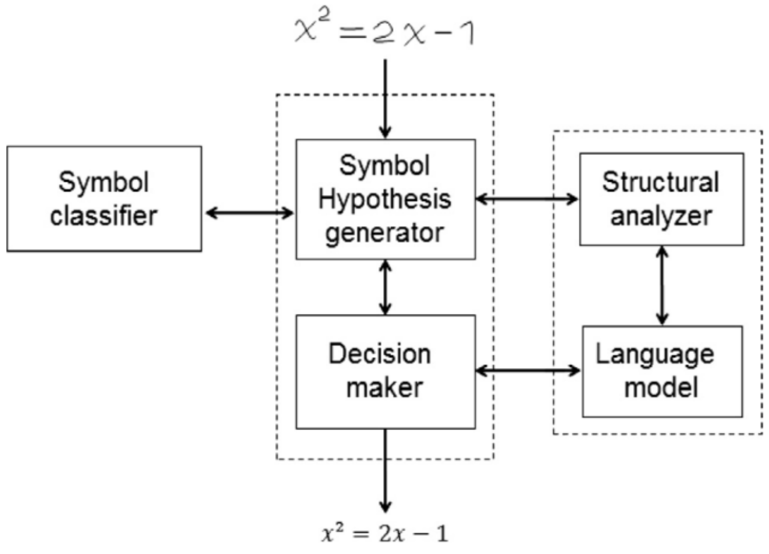


Figure 2.10 – The system architecture proposed in [Awal et al., 2014]. Extracted from [Awal et al., 2014].

dynamic programming algorithm provides a number of segmentation hypotheses. It allows grouping strokes which are not consecutive in time. Then they consider a symbol classifier with a reject capacity in order to deal with the invalid hypotheses proposed by the previous hypothesis generator. The structural costs are computed with Gaussian models which are learned from a training data set. The spatial information used are baseline position (y) and x-height (h) of one symbol or sub-expression hypothesis. The language model is defined by a combination of two 1-D grammars (horizontal and vertical). The production rules are applied successively until reaching elementary symbols, and then a bottom-up parse (CYK) is applied to construct the relational tree of the expression. Finally, the decision maker selects the set of hypotheses that minimizes the global cost function.

A fuzzy Relational Context-Free Grammar (r-CFG) and an associated top-down parsing algorithm are proposed in [MacLean and Labahn, 2013]. Fuzzy r-CFGs explicitly model the recognition process as a fuzzy relation between concrete inputs and abstract expressions. The production rules defined in this grammar have the form of: $A_0 \overset{r}{\Rightarrow} A_1 A_2 \cdots A_k$, where A_0 belongs to non-terminals and $A_1, \cdots, A_k$ belong to terminals. r denotes a relation between the elements $A_1, \cdots, A_k$. They use five binary spatial relations: $\nearrow$, $\rightarrow$, $\searrow$, $\downarrow$, $\odot$. The arrows indicate a general writing direction, while $\odot$ denotes containment (as in notations like $\sqrt{x}$, for instance). Figure 2.11 presents a simple example of this grammar. The parsing algorithm used

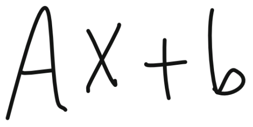
$$\begin{aligned}
\texttt{[ST]} &\Rightarrow \texttt{[ADD]} \mid \texttt{[TRM]}\\
\texttt{[ADD]} &\overset{\rightarrow}{\Rightarrow} \texttt{[TRM]} + \texttt{[ST]}\\
\texttt{[TRM]} &\Rightarrow \texttt{[MUL]} \mid \texttt{[SUP]} \mid \texttt{[CHR]}\\
\texttt{[MUL]} &\overset{\rightarrow}{\Rightarrow} \texttt{[SUP]}\,\texttt{[TRM]} \mid \texttt{[CHR]}\,\texttt{[TRM]}\\
\texttt{[SUP]} &\overset{\nearrow}{\Rightarrow} \texttt{[CHR]}\,\texttt{[ST]}\\
\texttt{[CHR]} &\Rightarrow \texttt{[VAR]} \mid \texttt{[NUM]}\\
\texttt{[VAR]} &\Rightarrow a \mid b \mid \cdots \mid z\\
\texttt{[NUM]} &\Rightarrow 0 \mid 1 \mid \cdots \mid 9
\end{aligned}$$

Figure 2.11 – A simple example of Fuzzy r-CFG. Extracted from [MacLean and Labahn, 2013].

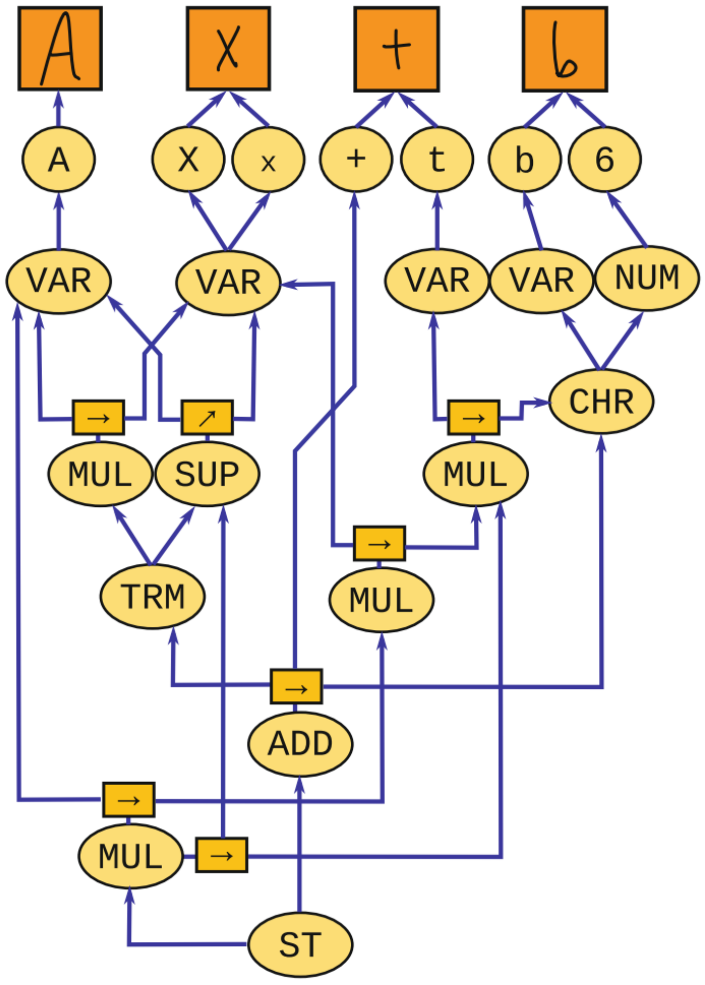
in this work is a tabular variant of Unger's method for CFG parsing [Unger, 1968]. This process is divided into two steps: forest construction, in which a shared parse forest is created from the start non-terminal to the leafs that represents all recognizable parses of the input, and tree extraction, in which individual parse trees are extracted from the forest in decreasing order of membership grade. Figure 2.12 show an handwritten expression and a shared parse forest of it representing some possible interpretations.

In [Álvaro et al., 2016], they define the statistical framework of a model based on Two-Dimensional Probabilistic Context-Free Grammars (2D-PCFGs) and its associated parsing algorithm. The authors also regard the problem of mathematical expression recognition as obtaining the most likely parse tree given a sequence of strokes. To achieve this goal, two probabilities are required, symbol likelihood and structural probability. Due to the fact that only strokes that are close together will form a mathematical symbol, a symbol likelihood model is proposed based on spatial and geometric information. Two concepts (visibility and closeness) describing the geometric and spatial relations between strokes are used in this work to characterize a set of possible segmentation hypotheses. Next, a BLSTM-RNN are used to calculate the probability that a certain segmentation hypothesis represents a math symbol. BLSTM possesses the ability to access context information over long periods of time from both past and future and is one of the state of the art models. With regard to the structural probability, both the probabilities of the rules of the grammar and a spatial relationship model which provides the probability $p(r|BC)$ that two sub-problems B and C are arranged according to spatial relationship r are required. In order to train a statistical classifier, given two regions B and C, they define nine geometric features based on their bounding boxes (Figure 2.13). Then these nine features are rewrote as the feature vector $h(B, C)$ representing a spatial relationship. Next, a GMM is trained with the labeled feature vector such that the probability of the spatial relationship model can be computed as the posterior probability provided by the GMM for class r. Finally, they define a CYK-based algorithm for 2D-PCFGs in the statistical framework.

Unlike the former described solutions which are based on string grammar, in [Julca-Aguilar, 2016], the authors model the recognition problem as a graph parsing problem. A graph grammar model for mathematical expressions and a graph parsing technique that integrates symbol and structure level information are proposed in this work. The recognition process is illustrated in Figure 2.14. Two main components are involved in this process: (1) hypotheses graph generator and (2) graph parser. The hypotheses graph generator builds a graph that defines the search space of the parsing algorithm and the graph parser does the parsing itself. In the hypotheses graph, vertices represent symbol hypotheses and edges represent relations



(a)



(b)

Figure 2.12 – (a) An input handwritten expression; (b) a shared parse forest of (a) considering the grammar depicted in Figure 2.11. Extracted from [MacLean and Labahn, 2013]

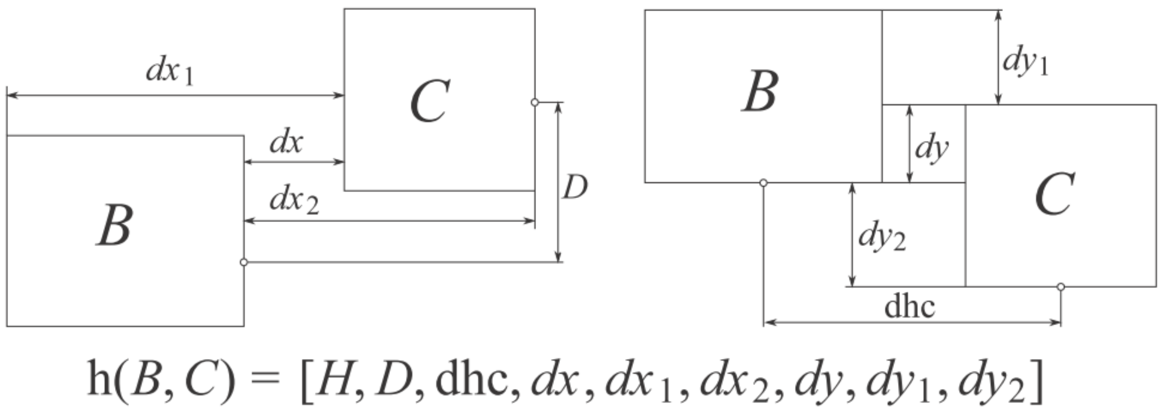

dx_1
dx
dx_2
D
dy_1
dy
dy_2
dhc
B
C
$h(B,C) = [H, D, \mathrm{dhc}, dx, dx_1, dx_2, dy, dy_1, dy_2]$

Figure 2.13 – Geometric features for classifying the spatial relationship between regions B and C. Extracted from [Álvaro et al., 2016]

between symbols. The labels associated to symbols and relations indicate their most likely interpretations. Of course, these labels are the outputs of symbol classifier and relation classifier. The graph parser uses the hypotheses graph and the graph grammar to generate first a parse forest consisting of several parse trees, each one representing an interpretation of the input strokes as a mathematical expression, and then extracts a best tree among the forest as the final recognition result. In the proposed graph grammar, production rules have the form of $A \rightarrow B$, defining the replacement of a graph by another graph. With regard to the parsing technique, they propose an algorithm based on the Unger's algorithm which is used for parsing strings [Unger, 1968]. The algorithm presented in this work is a top-down approach, starting from the top vertex (root) to the bottom vertices.

2.2.3 End-to-end neural network based solutions

In [Deng et al., 2016], the proposed model *WYGIWYS* (what you get is what you see) is an extension of the attention-based encoder-decoder model. The structure of *WYGIWYS* is shown in Figure 2.15. As can be seen, given an input image, a Convolutional Neural Network (CNN) is applied first to extract image features. Then, for each row in the feature map, they use an Recurrent Neural Network (RNN) encoder to re-encodes it expecting to catch the sequential information. Next, the encoded features are decoded by an RNN decoder with a visual attention mechanism to generate the final outputs. In parallel to the work of [Deng et al., 2016], [Zhang et al., 2017] also use the attention based encoder-decoder framework to translate MEs into LaTeX notations. Compared to the recent integrated solutions, the end-to-end neural network based solutions require no large amount of manual work for defining grammars or a high computational complexity for grammar parsing process, and achieve the state of the art recognition results. However, considering we are given trace information in the online case, despite the final LaTeX string, it is necessary to decide a label for each stroke. This alignment is not available now in end-to-end systems.

2.2.4 Discussion

In this section, we first introduce the development of mathematical expression recognition in general, and then put emphasis on the more recent proposed solutions. Instead of analyzing the advantages and disadvantages of the existing approaches consisting of variable grammars and their associated parsing techniques, the aim of this section is to provide a comparison to the new architectures proposed in this thesis. In spite of considerable different methods related to the three sub-tasks (symbol segmentation, symbol recognition and structural analysis), and variable grammars and parsing techniques, the key idea behind these

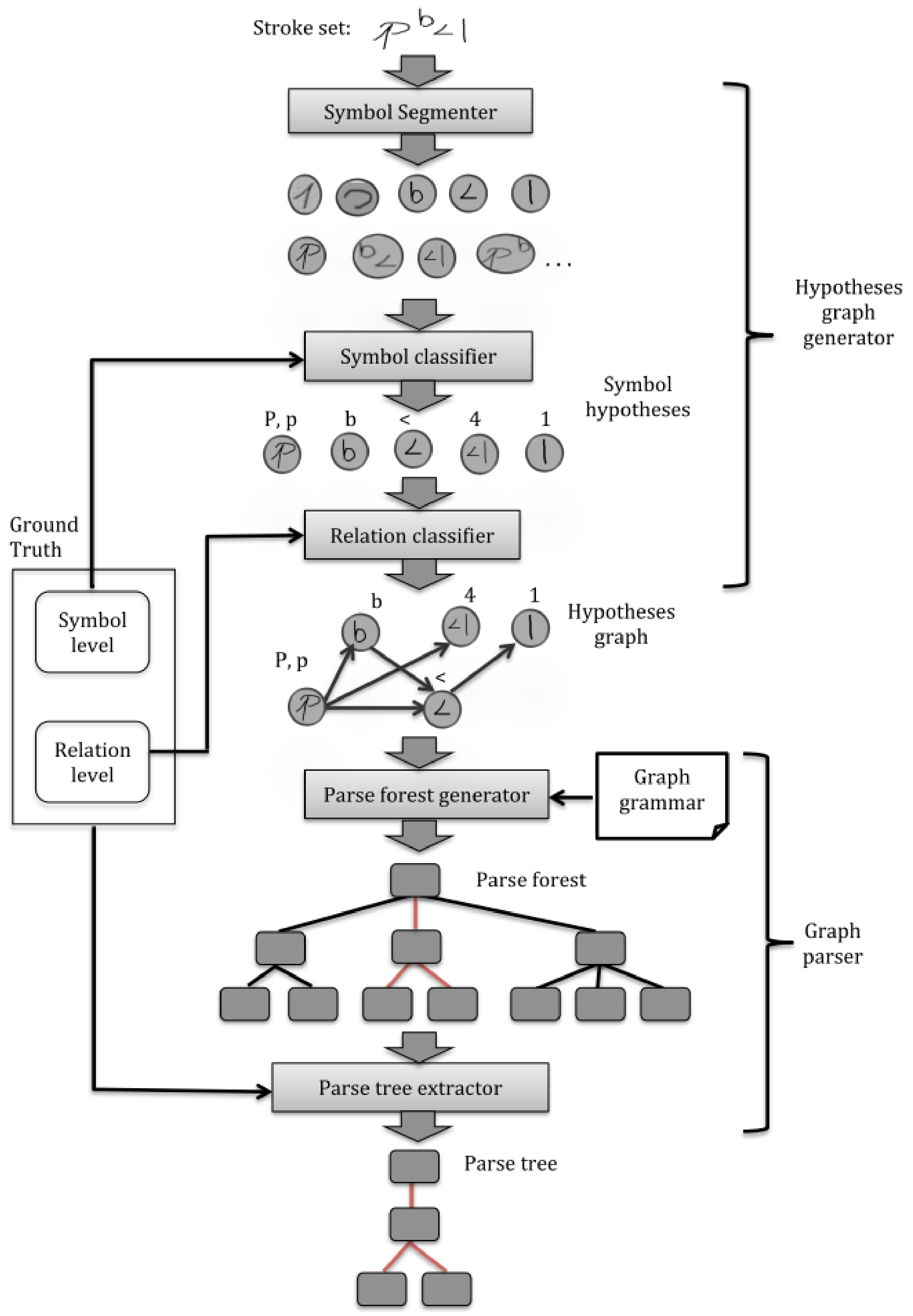


Figure 2.14 – Achitecture of the recognition system proposed in [Julca-Aguilar, 2016]. Extracted from [Julca-Aguilar, 2016]

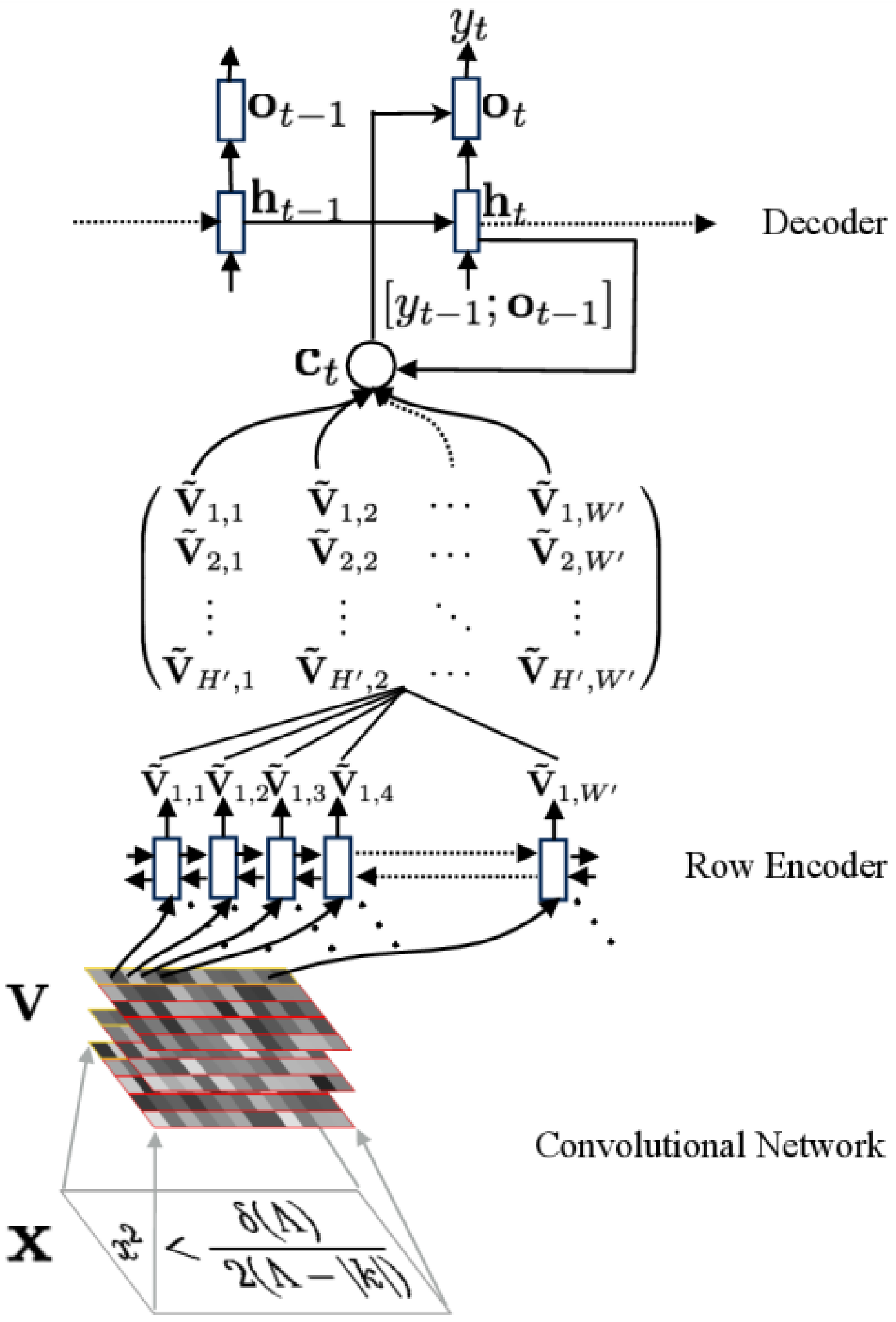


Figure 2.15 – Network architecture of *WYGIWYS*. Extracted from [Deng et al., 2016]

integrated techniques is relying on explicit grammar rules to solve the ambiguity in symbol recognition and relation recognition. In other words, the existing solutions take into account contextual or global information generally with the help of a grammar. However, using either string or graph grammar, a large amount of manual work is needed for defining grammars and a high computational complexity for grammar parsing process.

BLSTM neural network is able to model the dependency in a sequence over indefinite time gaps, overcoming the short-term memory of classical recurrent neural networks. Due to this ability, BLSTM achieved great success in sequence labeling tasks, such as text and speech recognition. Instead of using grammar parsing technique, the new architectures proposed in this thesis will include contextual information with bidirectional long short-term memory. In [Álvaro et al., 2016], it has been used an elementary function to recognize symbols or to control segmentation, which is itself included in an overall complex system. The goal of our work is to develop a new architecture where a recurrent neural network is the backbone of the solution.

In next chapter, we will introduce how the advanced neural network take the contextual information into consideration for the problem of sequence labeling.

3

Sequence labeling with recurrent neural networks

This chapter will be focused on sequence labeling using recurrent neural networks, which is the foundation of our work. Firstly, the concept of sequence labeling will be introduced in Section 3.1. We explain the goal of this task. Next, Section 3.2 introduces the classical structure of recurrent neural network. The property of this network is that it can memorize contextual information but the range of the information which could be accessed is quite limited. Subsequently, in Section 3.3 long short-term memory is presented. This architecture is provided with the ability of accessing information over long periods of time. Finally, we introduce how to apply recurrent neural network for the task of sequence labeling, including the existing problems and the solutions to solve them, i.e. the connectionist temporal classification technique.

In this chapter, considerable amount of variables and formulas are involved in order to clearly describe the content, likewise to extend easily the algorithms in later chapters. We use here the same notations as in [Graves et al., 2012]. In fact, this chapter is a short version of Alex Graves' book «Supervised sequence labeling with recurrent neural networks». We use the same figures and similar outline to introduce this entire framework. Since the architecture of BLSTM and CTC is the backbone of our solution, thus we take a whole chapter to elaborate this topology to help to understand our work.

3.1 Sequence labeling

In machine learning, the term 'sequence labeling' encompasses all tasks where sequences of data are transcribed with sequences of discrete labels [Graves et al., 2012]. Well known examples include handwriting and speech recognition (Figure 3.1), gesture recognition and protein secondary structure. In this thesis, we only consider supervised sequence labeling cases in which the ground-truth is provided during the training process.

The goal of sequence labeling is to transcribe sequences of input data into sequences of labels, each label coming from a fixed alphabet. For example looking at the top row of Figure 3.1, we would like to assign the sequence "FOREIGN MINISTER" of which each label is from English alphabet, to the input signal on the left side. Suppose that X denotes a input sequence and l is the corresponding ground truth, being a sequence of labels, the set of training examples could be referred as $Tra = \{(X, l)\}$. The task is to use Tra to train a sequence labeling algorithm to label each input sequence in a test data set, as accurately as possible. In fact when people try to recognize a handwriting or speech signal, we focus on not only local input signal, but also a global, contextual information to help the transcription process. Thus, we hope the

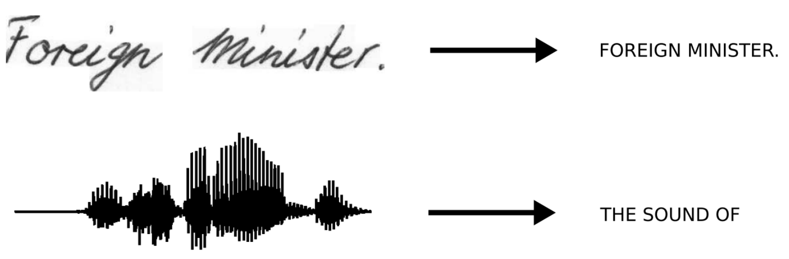


Figure 3.1 – Illustration of sequence labeling task with the examples of handwriting (top) and speech (bottom) recognition. Input signals is shown on the left side while the ground truth is on the right. Extracted from [Graves et al., 2012].

sequence labeling algorithm could have the ability also to take advantage of contextual information.

3.2 Recurrent neural networks

Artificial Neural Networks (ANNs) are computing systems inspired by the biological neural networks [Jain et al., 1996]. It is hoped that such systems could possess the ability to learn to do tasks by considering some given examples. An ANN is a network of small units, joined to each other by weighted connections. Whether connections form cycles or not, usually we can divide ANNs into two classes: ANNs without cycles are referred to as Feed-forward Neural Networks (FNNs); ANNs with cycles, are referred to as feedback, recurrent neural networks (RNNs). The cyclical connections could model the dependency between past and future, therefore RNNs possess the ability to memorize while FNNs do not have memory capability.

In this section, we will focus on recurrent networks with cyclical connections. Thanks to RNN's memory capability, it is suitable for sequence labeling task where the contextual information plays a key role. Many varieties of RNN were proposed, such as Elman networks, Jordan networks, time delay neural networks and echo state networks [Graves et al., 2012]. We introduce here a simple RNN architecture containing only a single, self connected hidden layer (Figure 3.3).

3.2.1 Topology

In order to better understand the mechanism of RNNs, we first provide a short introduction to Multilayer Perceptron (MLP) [Rumelhart et al., 1985, Werbos, 1988, Bishop, 1995] which is the most widely used form of FNNs. As illustrated in Figure 3.2, a MLP has an input layer, one or more hidden layers and an output layer. The S-shaped curves in the hidden and output layers indicate the application of 'sigmoidal' nonlinear activation functions. The number of units in the input layer is equal to the length of feature vector. Both the number of units in the output layer and the choice of output activation function depend on the task the network is applied to. When dealing with binary classification tasks, the standard configuration is a single unit with a logistic sigmoid activation. For classification problems with $K > 2$ classes, usually we have K output units with the soft-max function. Since there is no connection from past to future or future to past, MLP depends only on the current input to compute the output and therefore is not suitable for sequence labeling.

Unlike the feed forward network architecture, in a neural network with cyclical connections presented in Figure 3.3, the connections from the hidden layer to itself (red) could model the dependency between past and future. However, the dependencies between different time-steps can not be seen clearly in this figure. Thus, we unfold the network along the input sequence to visualize them in Figure 3.4. Different with Figure 3.2 and 3.3 where each node is a single unit, here each node represents a layer of network units

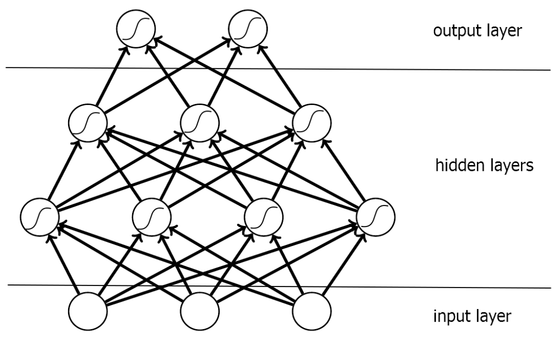


Figure 3.2 – A multilayer perceptron.

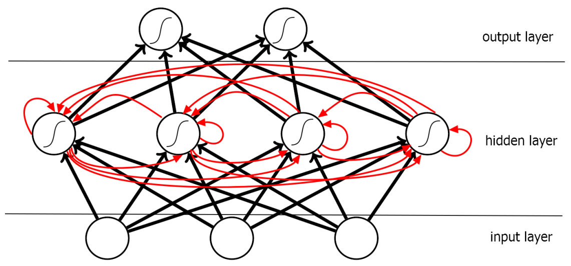


Figure 3.3 – A recurrent neural network. The recurrent connections are highlighted with red color.

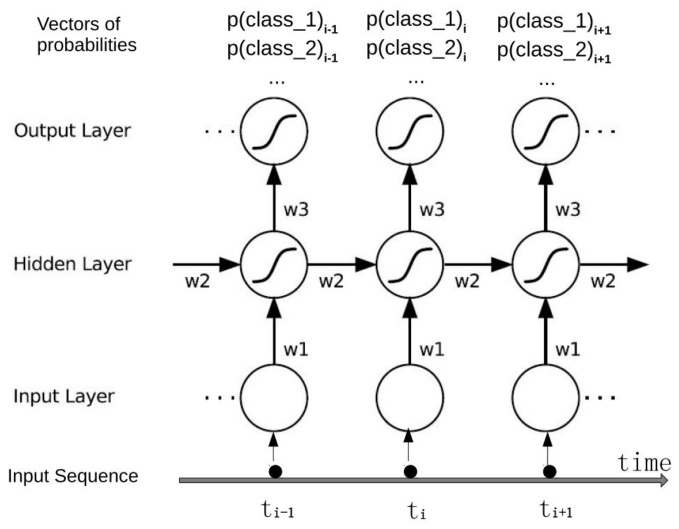


Figure 3.4 – An unfolded recurrent network.

at a single time-step. The input at each time step is a vector of features; the output at each time step is a vector of probabilities regarding to different classes. With the connections weighted by 'w1' from the input layer to hidden layer, the current input flows to the current hidden layer; with the connections weighted by 'w2' from the hidden layer to itself, the information flows from the the hidden layer at $t-1$ to the hidden layer at t; with the connections weighted by 'w3' from the hidden layer to the output layer, the activation flows from the hidden layer to the output layer. Note that 'w1', 'w2' and 'w3' represent vectors of weights instead of single weight values, and they are reused for each time-step.

3.2.2 Forward pass

The input data flow from the input layer to hidden layer; the output activation of the hidden layer at $t-1$ flows to the hidden layer at t; the hidden layer sums up the information from two sources; finally the summed and processed information flows to the output layer. This process is referred to as the **forward pass** of RNN. Suppose that an RNN has I input units, H hidden units, and K output units, let w_{ij} denote the weight of the connection from unit i to unit j, a_j^t and b_j^t represent the network input activation to unit j and the output activation of unit j at time t respectively. Specifically, we use use x_i^t to denote the input i value at time t. Considering an input sequence X of length T, the network input activation to the hidden units could be computed like:

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \tag{3.1}$$

In this equation, we can see clearly that the activation arriving at the hidden layer comes from two sources: (1) the current input layer through the 'w1' connections; (2) the hidden layer of previous time step through the 'w2' connections. The size of 'w1' and 'w2' are respectively $size(w1) = I \times H + 1(bias)$ and $size(w2) = H \times H$. Then, the activation function θ_h is applied:

$$b_h^t = \theta_h(a_h^t) \tag{3.2}$$

We calculate a_h^t and therefore b_h^t from $t=1$ to T. This is a recursive process where a initial configuration is required of course. In this thesis, the initial value $b_{h'}^0$ is always set to 0. Now, we consider propagating the hidden layer output activation b_h^t to the output layer. The activation arriving at the output units can be calculated as following:

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t \tag{3.3}$$

The size of 'w3' is $size(w3) = H \times K$. Then applying the activation function θ_k, we get the output activation b_k^t of the output layer unit k at time t. We use a a special name y_k^t to represent it:

$$y_k^t = \theta_k(a_k^t) \tag{3.4}$$

We introduce the definition of the loss function in Section 3.4.

3.2.3 Backward pass

With the loss function, we could compute the distance between the network outputs and the ground truths. The aim of **backward pass** is to minimize the distance to train an effective neural network. The widely used solution is *gradient descent* of which the idea is to first calculate the derivative of the loss function with respect to each weight and then adjust the weights in the direction of negative slope to minimize the loss function [Graves et al., 2012].

To compute the derivative of the loss function with respect to each weight in the network, the common technique used is known as Back Propagation (BP) [Rumelhart et al., 1985, Williams and Zipser, 1995, Werbos, 1988]. As there are recurrent connections in RNNs, researchers designed the special algorithms to calculate weight derivatives efficiently for RNNs, two well known methods being Real Time Recurrent Learning (RTRL) [Robinson and Fallside, 1987] and Back Propagation Through Time (BPTT) [Williams and Zipser, 1995] [Werbos, 1990]. Like Alex Graves, we introduce BPTT only as it is both conceptually simpler and more efficient in computation time.

We define

$$\delta_j^t = \frac{\partial L}{\partial a_j^t} \tag{3.5}$$

Thus the partial derivatives of the loss function L with respect to the inputs of the output units a_k^t is

$$\delta_k^t = \frac{\partial L}{\partial a_k^t} = \sum_{k'=1}^{K} \frac{\partial L}{\partial y_{k'}^t} \frac{\partial y_{k'}^t}{\partial a_k^t} \tag{3.6}$$

Afterwards, the error will be back propagated to the hidden layer. Note that the loss function depends on the activation of the hidden layer not only through its influence on the output layer, but also through its influence on the hidden layer at the next time-step. Thus,

$$\delta_h^t = \frac{\partial L}{\partial a_h^t} = \frac{\partial L}{\partial b_h^t}\frac{\partial b_h^t}{\partial a_h^t} = \frac{\partial b_h^t}{\partial a_h^t}\Big(\sum_{k=1}^{K} \frac{\partial L}{\partial a_k^t}\frac{\partial a_k^t}{\partial b_h^t} + \sum_{h'=1}^{H} \frac{\partial L}{\partial a_{h'}^{t+1}}\frac{\partial a_{h'}^{t+1}}{\partial b_h^t}\Big) \tag{3.7}$$

$$\delta_h^t = \theta'_h a_h^t \left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'} \right) \tag{3.8}$$

δ_h^t terms can be calculated recursively from T to 1. Of course this requires the initial value δ_h^{T+1} to be set. As there is no error coming from beyond the end of the sequence, $\delta_h^{T+1} = 0 \; \forall h$. Finally, noticing that the same weights are reused at every time-step, we sum over the whole sequence to get the derivatives with respect to the network weights

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t \tag{3.9}$$

The last step is to adjust the weights based on the derivatives we have computed above. It is an easy procedure and we do not discuss it here.

3.2.4 Bidirectional networks

The RNNs we have discussed only possess the ability to access the information from past, not the future. In fact, future information is important to sequence labeling task as well as the past context. For example when we see the left bracket '(' in the handwritten expression $2(a+b)$, it seems easy to answer '1', 'l' or '(' if only focusing on the signal on the left side of '('. But if we consider the signal on the right side also, the answer is straightforward, being '(' of course. An elegant solution to access context from both directions is Bidirectional Recurrent Neural Networks (BRNNs) (BRNNs) [Schuster and Paliwal, 1997, Schuster, 1999, Baldi et al., 1999].

Figure 3.5 shows an unfolded bidirectional network. As we can see, there are 2 separate recurrent hidden layers, forward and backward, each of them process the input sequence from one direction. No information flows between the forward and backward hidden layers and these two layers are both connected to the same output layer. With the bidirectional structure, we could use the complete past and future context to help recognizing each point in the input sequence.

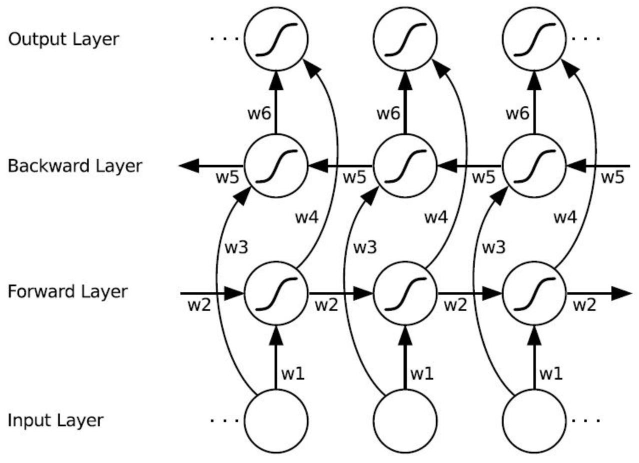


Figure 3.5 – An unfolded bidirectional network. Extracted from [Graves et al., 2012].

3.3 Long short-term memory (LSTM)

In Section 3.2, we discussed RNNs which have the ability to access contextual information from one direction and BRNNs which have the ability to visit bidirectional contextual information. Due to their memory capability, lots of applications are available in sequence labeling tasks. However, there is a problem that the range of context that can be in practice accessed is quite limited. The influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections [Graves et al., 2012]. This effect is often referred to in the literature as the vanishing gradient problem [Hochreiter et al., 2001, Bengio et al., 1994]. To address this problem, many methods were proposed such as simulated annealing and discrete error propagation [Bengio et al., 1994], explicitly introduced time delays [Lang et al., 1990, Lin et al., 1996, Giles et al.] or time constants [Mozer, 1992], and hierarchical sequence compression [Schmidhuber, 1992]. In this section, we will focus on Long Short-Term Memory (LSTM) architecture [Hochreiter and Schmidhuber, 1997].

3.3.1 Topology

We replace the summation unit in the hidden layer of a standard RNN with memory block (Figure 3.6), generating an LSTM network. There are three gates (input gate, forget gate and output gate) and one or more cells in a memory block. Figure 3.6 shows a LSTM memory block with one cell. We list below the activation arriving at three gates at time t:

Input gate: the current input, the activation of hidden layer at time $t-1$, the cell state at time $t-1$
Forget gate: the current input, the activation of hidden layer at time $t-1$, the cell state at time $t-1$
Output gate: the current input, the activation of hidden layer at time $t-1$, the current cell state

The connections shown by dashed lines from the cell to three gates are named as 'peephole' connections which are the only weighted connections inside the memory block. Just because of the three 'peephole's, the cell state is accessible to the three gates. These three gates sum up the information from inside and outside the block with different weights and then apply gate activation function 'f', usually the logistic sigmoid. Thus, the gate activation are between 0 (gate closed) and 1 (gate open). We present below how these three gates control the cell via multiplications (small black circles):

Input gate: the input gate multiplies the input of the cell. The input gate activation decides how much information the cell could receive from the current input layer, 0 representing no information and 1 repre-

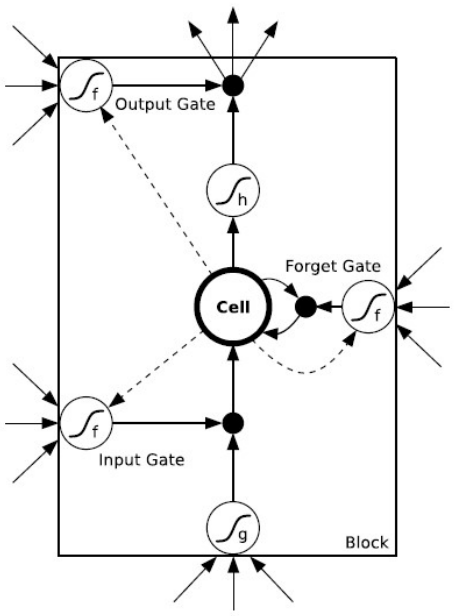


Figure 3.6 – LSTM memory block with one cell. Extracted from [Graves et al., 2012].

senting all the information.
Forget gate: the forget gate multiplies the cell's previous state. The forget gate activation decides how much context should the cell memorize from its previous state, 0 representing forgetting all and 1 representing memorizing all.
Output gate: the output gate multiplies the output of the cell. It controls to which extent the cell will output its state, 0 representing nothing and 1 representing all.

The cell input and output activation functions ('g' and 'h') are usually tanh or logistic sigmoid, though in some cases 'h' is the identity function [Graves et al., 2012]. Output gate controls to which extent the cell will output its state, and it is the only outputs from the block to the rest of the network.

As we discussed, the three control gates could allow the cell to receive, memorize and output information selectively, thereby easing the vanishing gradient problem. For example the cell could memorize totally the input at first point as long as the forget gates are open and the input gates are closed at the following time steps.

3.3.2 Forward pass

As in [Graves et al., 2012], we only present the equations for a single memory block since it is just a repeated calculation for multiple blocks. Let w_{ij} denote the weight of the connection from unit i to unit j, a_j^t and b_j^t represent the network input activation to unit j and the output activation of unit j at time t respectively. Specifically, we use use x_i^t to denote the input i value at time t. Considering a recurrent network with I input units, K output units and H hidden units, the subscripts ς, ϕ, ω represent the input, forget and output gate and the subscript c represents one of the C cells. Thus, the connections from the input layer to the three gates are weighted by $w_{i\varsigma}$, $w_{i\phi}$, $w_{i\omega}$ respectively; the recurrent connections to the three gates are weighted by $w_{h\varsigma}$, $w_{h\phi}$, $w_{h\omega}$; the peep-hole 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^t is the state of cell c at time t. We use f to denote the activation function of the gates, and g and h to denote respectively the cell input and output activation functions. b_c^t is the only output from the block to the rest of the network. As with the standard RNN, the forward pass is a recursive calculation by starting at $t = 1$. All the related initial values are set to 0.

Equations are given below:

Input gates

$$a_\varsigma^t = \sum_{i=1}^{I} w_{i\varsigma} x_i^t + \sum_{h=1}^{H} w_{h\varsigma} b_h^{t-1} + \sum_{c=1}^{C} w_{c\varsigma} s_c^{t-1} \tag{3.10}$$

$$b_\varsigma^t = f(a_\varsigma^t) \tag{3.11}$$

Forget gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{3.12}$$

$$b_\phi^t = f(a_\phi^t) \tag{3.13}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{3.14}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\varsigma^t g(a_c^t) \tag{3.15}$$

Output gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^{t} \tag{3.16}$$

$$b_\omega^t = f(a_\omega^t) \tag{3.17}$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{3.18}$$

3.3.3 Backward pass

As can be seen in Figure 3.6, a memory block has 4 interfaces receiving inputs from outside the block, 3 gates and one cell. Considering the hidden layer, the total number of input interfaces is defined as G. For the memory block consisting only one cell, G is equal to $4H$. We recall Equation 3.5

$$\delta_j^t = \frac{\partial L}{\partial a_j^t} \tag{3.19}$$

Furthermore, define

$$\epsilon_c^t = \frac{\partial L}{\partial b_c^t} \qquad \epsilon_s^t = \frac{\partial L}{\partial s_c^t} \tag{3.20}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{g=1}^{G} w_{cg} \delta_g^{t+1} \tag{3.21}$$

As b_c^t is propagated to the output layer and the hidden layer of next time step in the forward pass, when computing ϵ_c^t, it is natural to receive the derivatives from both the output layer and the next hidden layer. G is introduced for the convenience of representation.

Output gates

$$\delta_w^t = f'(a_w^t) \sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \tag{3.22}$$

States

$$\epsilon_s^t = b_w^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\varsigma}\delta_\varsigma^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{3.23}$$

Cells

$$\delta_c^t = b_\varsigma^t g'(a_c^t)\epsilon_s^t \tag{3.24}$$

Forget gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{3.25}$$

Input gates

$$\delta_\varsigma^t = f'(a_\varsigma^t) \sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{3.26}$$

3.3.4 Variants

There exists many variants of the basic LSTM architecture. Globally, they can be divided into chain-structured LSTM and non-chain-structured LSTM.

Bidirectional LSTM

Replacing the hidden layer units in BRNN with LSTM memory blocks generates Bidirectional LSTM [Graves and Schmidhuber, 2005]. LSTM network processes the input sequence from past to future while Bidirectional LSTM, 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 [Graves et al., 2013] can be created by stacking multiple BLSTM layers on top of each other in order to get higher level representation of the input data. As illustrated in Figure 3.7, 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 network topology described thus far is that they only allow for sequential information propagation (as shown in Figure 3.8a) 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 using n recurrent connections (one for each of the cell's previous states along every dimension) with n forget gates. It is named Multidimensional LSTM (MDLSTM) dedicated to the graph structure of an n-dimensional grid such as images [Graves et al., 2012]. In [Tai et al., 2015], the basic LSTM architecture was extend to tree structures, the Child-sum Tree-LSTM and the N-ary Tree-LSTM, allowing for richer network topology (Figure 3.8b) where each unit is able to incorporate information from multiple child units. In parallel to the work in [Tai et al., 2015], [Zhu et al., 2015] explores the similar idea. The DAG-structured LSTM was proposed for semantic compositionality [Zhu et al., 2016].

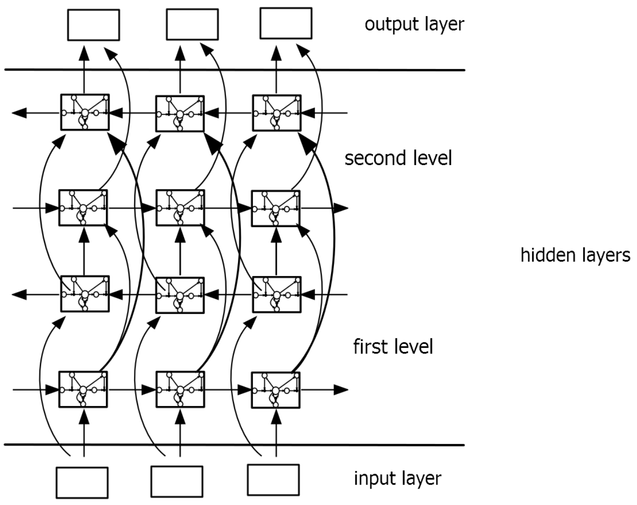


Figure 3.7 – A deep bidirectional LSTM network with two hidden levels.

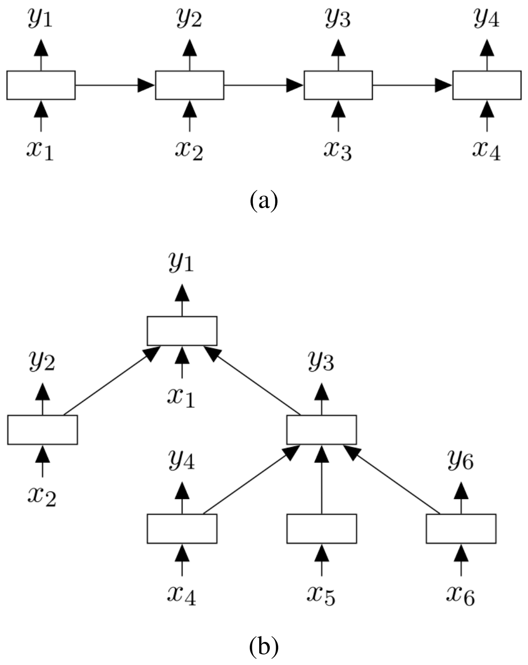


Figure 3.8 – (a) A chain-structured LSTM network; (b) A tree-structured LSTM network with arbitrary branching factor. Extracted from [Tai et al., 2015].

In later chapter, we will extend the chain-structured BLSTM to tree-based BLSTM which is similar to the above mentioned work, and apply this new network model for online math expression recognition.

3.4 Connectionist temporal classification (CTC)

RNNs' memory capability greatly meet the sequence labeling tasks where the context is quite important. To apply this recurrent network into sequence labeling, at least a loss function should be defined for the training process. In the typical frame wise training method, we need to know the ground truth label for each time step to compute the errors which means pre-segmented training data is required. 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 solve these two points. 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. We introduce CTC briefly here; for a more detailed description, refer to A. Graves' book [Graves et al., 2012].

3.4.1 From outputs to labelings

CTC consists of a soft max output layer with one more unit ($blank$) than there are labels in alphabet. Suppose the alphabet is A ($|A| = N$), the new extended alphabet is A' which is equal to $A \cup [blank]$. Let y_k^t denote the probability of outputting the k label of A' at the t time step given the input sequence X of length T, where k is from 1 to $N+1$ and t is from 1 to T. Let A'^T denote the set of sequences over A' with length T and any sequence $\pi \in A'^T$ is referred to as a path. Then, assuming the output probabilities at each time-step to be independent of those at other time-steps, the probability of outputting a sequence π would be:

$$p(\pi|X) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{3.27}$$

The next step is from π to get the real possible labeling of X. A many-to-one function $F : A'^T \to A^{\leq T}$ is defined from the set of paths onto the set of possible labeling of X to do this task. Specifically, first remove the repeated labels and then the blanks (–) from the paths. For example considering an input sequence of length 11, two possible paths could be $cc--aaa-tt-, c---aa--ttt$. The mapping function works like: $F(cc--aaa-tt-) = F(c---aa--ttt) = cat$. Since the paths are mutually exclusive, the probability of a labeling sequence $l \in A^{\leq T}$ can be calculated by summing the probabilities of all the paths mapped onto it by F :

$$p(l|X) = \sum_{\pi \in F^{-1}(l)} p(\pi|X) \tag{3.28}$$

3.4.2 Forward-backward algorithm

In section 3.4.1, we defined the probability $p(l|X)$ as the sum of the probabilities of all the paths mapped onto l. The calculation seems to be problematic because the number of paths grows exponentially with the length of the input sequence. Fortunately it can be solved with a dynamic-programming algorithm similar to the forward-backward algorithm for Hidden Markov Model (HMM) [Bourlard and Morgan, 2012].

Consider a modified label sequence l' with blanks added to the beginning and the end of l, and inserted between every pair of consecutive labels. Suppose that the length of l is U, apparently the length of l' is $U' = 2U + 1$. For a labeling l, let the forward variable $\alpha(t, u)$ denote the summed probability of all length t paths that are mapped by F onto the length $u/2$ prefix of l, and let the set $V(t, u)$ be equal to

$\{\pi \in A'^t : F(\pi) = l_{1:u/2}, \pi_t = l'_u\}$, where u is from 1 to U' and $u/2$ is rounded down to an integer value. Thus:

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y^i_{\pi_i} \tag{3.29}$$

All the possible paths mapped onto l start with either a blank (–) or the first label (l_1) of l, so we have the formulas below:

$$\alpha(1,1) = y^1_- \tag{3.30}$$

$$\alpha(1,2) = y^1_{l_1} \tag{3.31}$$

$$\alpha(1,u) = 0, \forall u > 2 \tag{3.32}$$

In fact, the forward variables at time t can be calculated recursively from those at time $t-1$.

$$\alpha(t,u) = y^t_{l'_u} \sum_{i=f(u)}^{u} \alpha(t-1,i), \forall t > 1 \tag{3.33}$$

where

$$f(u) = \begin{cases} u-1 & \text{if } l'_u = \text{blank or } l'_{u-2} = l'_u \\ u-2 & \text{otherwise} \end{cases} \tag{3.34}$$

Note that

$$\alpha(t,u) = 0, \forall u < U' - 2(T-t) - 1 \tag{3.35}$$

Given the above formulation, the probability of l can be expressed as the sum of the forward variables with and without the final blank at time T.

$$p(l|X) = \alpha(T,U') + \alpha(T,U'-1) \tag{3.36}$$

Figure 3.9 illustrates the CTC forward algorithm.

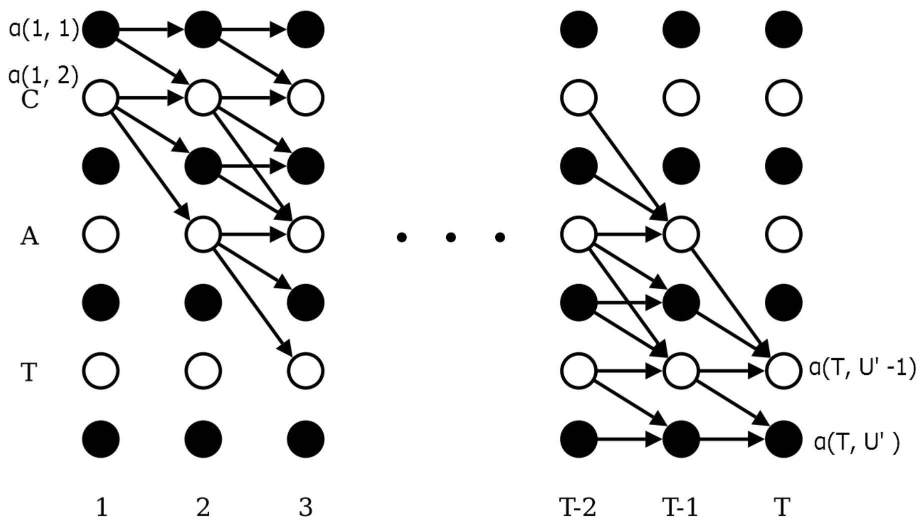


Figure 3.9 – Illustration of CTC forward algorithm. Blanks are represented with black circles and labels are white circles. Arrows indicate allowed transitions. Adapted from [Graves et al., 2012].

Similarly, we define the backward variable $\beta(t,u)$ as the summed probabilities of all paths starting at $t+1$ that complete l when appended to any path contributing to $\alpha(t,u)$. Let $W(t,u) = \{\pi \in A'^{T-t} : F(\hat{\pi} + \pi) = l, \forall \hat{\pi} \in V(t,u)\}$ denote the set of all paths starting at $t+1$ that complete l when appended to any path contributing to $\alpha(t,u)$. Thus:

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{\pi_i}^{t+i} \tag{3.37}$$

The formulas below are used for the initialization and recursive computation of $\beta(t,u)$:

$$\beta(T,U') = 1 \tag{3.38}$$

$$\beta(T,U'-1) = 1 \tag{3.39}$$

$$\beta(T,u) = 0, \forall u < U'-1 \tag{3.40}$$

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{l'_i}^{t+1} \tag{3.41}$$

where

$$g(u) = \begin{cases} u+1 & \text{if } l'_u = \text{blank or } l'_{u+2} = l'_u \\ u+2 & \text{otherwise} \end{cases} \tag{3.42}$$

Note that

$$\beta(t,u) = 0, \forall u > 2t \tag{3.43}$$

If we reverse the direction of the arrows in Figure 3.9, it comes to be an illustration of the CTC backward algorithm.

3.4.3 Loss function

The CTC loss function $L(S)$ is defined as the negative log probability of correctly labeling all the training examples in some training set S. Suppose that z is the ground truth labeling of the input sequence X, then:

$$L(S) = -\ln \prod_{(X,z)\in S} p(z|X) = -\sum_{(X,z)\in S} \ln p(z|X) \tag{3.44}$$

BLSTM networks can be trained to minimize the differentiable loss function $L(S)$ using any gradient-based optimization algorithm. The basic idea is to find the derivative of the loss function with respect to each of the network weights, then adjust the weights in the direction of the negative gradient.

The loss function for any training sample is defined as:

$$L(X,z) = -\ln p(z|X) \tag{3.45}$$

and therefore

$$L(S) = \sum_{(X,z)\in S} L(X,z) \tag{3.46}$$

The derivative of the loss function with respect to each network weight can be represented as:

$$\frac{\partial L(S)}{\partial w} = \sum_{(X,z)\in S} \frac{\partial L(X,z)}{\partial w} \tag{3.47}$$

The forward-backward algorithm introduced in Section 3.4.2 can be used to compute $L(X,z)$ and the gradient of it. We only provide the final formula in this thesis and the process of derivation can be found in [Graves et al., 2012].

$$L(X,z) = -\ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{3.48}$$

To find the gradient, the first step is to differentiate $L(X, z)$ with respect to the network outputs y_k^t:

$$\frac{\partial L(X,z)}{\partial y_k^t} = -\frac{1}{p(z|X)y_k^t} \sum_{u \in B(z,k)} \alpha(t,u)\beta(t,u) \tag{3.49}$$

where $B(z, k) = \{u : z'_u = k\}$ is the set of positions where label k occurs in z'. Then we continue to backpropagate the loss through the output layer:

$$\frac{\partial L(X,z)}{\partial a_k^t} = y_k^t - \frac{1}{p(z|X)} \sum_{u \in B(z,k)} \alpha(t,u)\beta(t,u) \tag{3.50}$$

and finally through the entire network during training.

3.4.4 Decoding

We discuss above how to train a RNN with CTC technique, and the next step is to label some unknown input sequence X in the test set with the trained model by choosing the most probable labeling l^* :

$$l^* = arg\max_l p(l|X) \tag{3.51}$$

The task of labeling unknown sequences is denoted as *decoding*, being a terminology coming from hidden Markov models (HMMs). In this section, we will introduce in brief several approximate methods that perform well in practice. Likewise, we refer the interested readers to [Graves et al., 2012] for the detailed description. We also design new decoding methods which are suitable to the tasks of this thesis in later chapters.

Best path decoding

Best path decoding is based on the assumption that the most probable path corresponds to the most probable labeling

$$l^* \approx F(\pi^*) \tag{3.52}$$

where $\pi^* = arg\ max_\pi p(\pi|X)$. It is simple to find π^*, just concatenating the most active outputs at each time-step. However best path decoding could lead to errors in some cases when a label is weakly predicted for several successive time-steps. Figure 3.10 illustrates one of the failed cases. In this simple case where there are just two time steps, the most probable path found with best path decoding is '$--$' with the probability of $0.42 = 0.7 * 0.6$, and therefore the final labeling is 'blank'. In fact, the summed probabilities of the paths corresponding to the labeling of 'A' is 0.58, greater than 0.42.

Prefix search decoding

Prefix search decoding is a best-first search through the tree of labelings, where the children of a given labeling are those that share it as a prefix. At each step the search extends the labeling whose children have the largest cumulative probability. As can be seen in Figure 3.11, there exist in this tree 2 types of nodes, *end* node ('e') and *extending* node. An *extending* node extends the prefix at its parent node and the number above it is the total probability of all labelings beginning with that prefix. An *end* node denotes that the labeling ends at its parent and the number above it is the probability of the single labeling ending at its parent. At each iteration, we explore the extending of the most probable remaining prefix. Search ends when a single labeling is more probable than any remaining prefix.

Prefix search decoding could find the most probable labeling with enough time. However the fact that the number of prefixes it must expand grows exponentially with the input sequence length, affects largely the feasibility of its application.

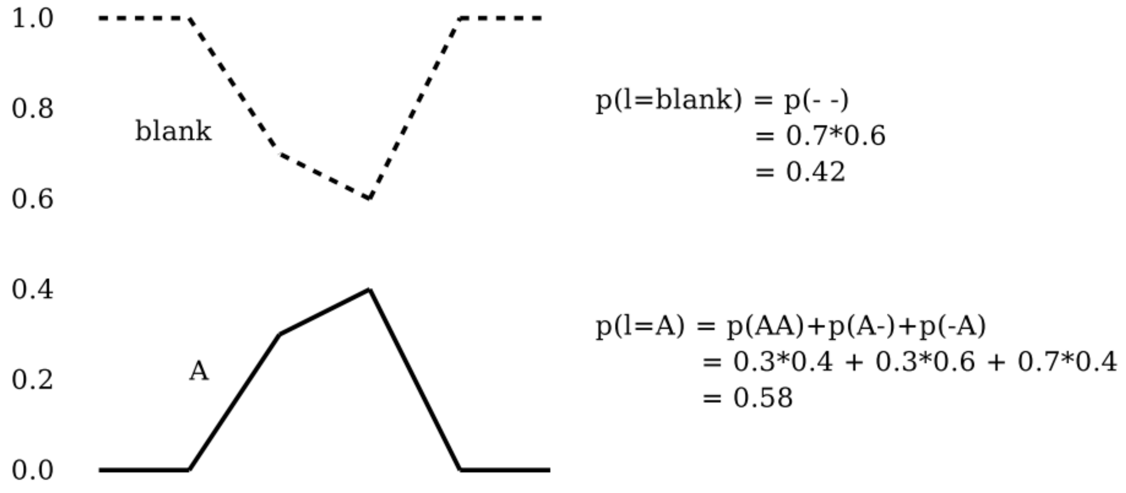


Figure 3.10 – Mistake incurred by best path decoding. Extracted from [Graves et al., 2012].

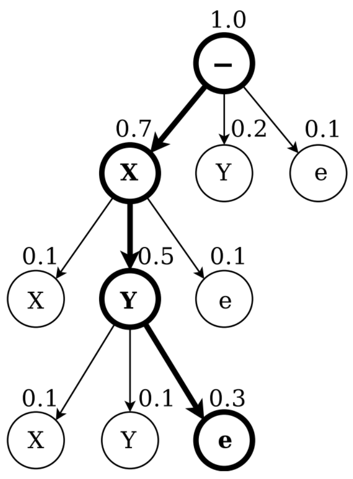


Figure 3.11 – Prefix search decoding on the alphabet {X, Y}. Extracted from [Graves et al., 2012].

Constrained decoding

Constrained decoding refers to the situation where we constrain the output labelings according to some predefined grammar. For example, in word recognition, the final transcriptions are usually required to form sequences of dictionary words. Here, we only consider single word decoding, which means all word-to-word transitions are forbidden.

With regard to single word recognition, if the number of words in the target sequence is fixed, one of the possible methods could be as following: considering an input sequence X, for each word wd in the dictionary, we firstly calculate the sum of the probabilities $p(wd|X)$ of all the paths π which can be mapped into wd using the forward-backward algorithm described in Section 3.4.2; then, assign X with the word holding the maximum probability.

II

Contributions

4 Mathematical expression recognition with single path

As well known, BLSTM network with a CTC output layer achieved great success in sequence labeling tasks, such as text and speeches recognition. This success is due to the LSTM's ability of capturing long-term dependency in a sequence and the effectiveness of CTC training method. In this chapter, we will explore the idea of using the sequence-structured BLSTM with a CTC stage to recognize 2-D handwritten mathematical expression (Figure 4.1). CTC allows the network to make label predictions at any point in the

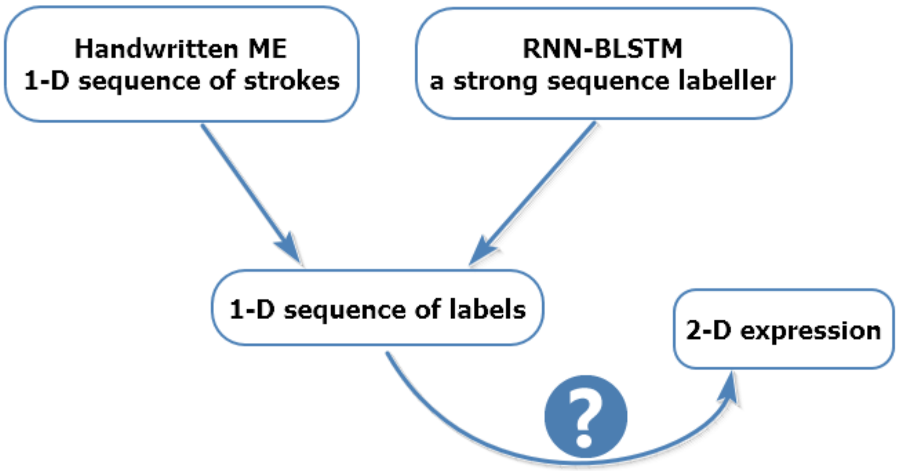


Figure 4.1 – Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME.

input sequence, so long as the overall sequence of labels is correct. It is not well suited for our cases in which a relatively precise alignment between the input and output is required. Thus, a local CTC methodology is proposed aiming to constrain the outputs to emit at least once or several times the same non-blank label in a given stroke.

This chapter will be organized as follows: Section 4.1 globally introduce the proposal that builds stroke label graph from a sequence of labels, along with the existing limitations in this stage. Then, the entire process of generating the sequence of labels with BLSTM and local CTC given the input is orderly presented in detail, including firstly feeding the inputs of BLSTM, then the training and recognition stages. The experiments and discussion are introduced in Section 4.3 and Section 4.4 respectively.

4.1 From single path to stroke label graph

This section will be focused on introducing the idea of building SLG from a single path. First, a classification of the degree of complexity of math expressions will be given to help understanding the different difficulties and the cases that could or could not be solved by the proposed approach.

4.1.1 Complexity of expressions

Expressions could be divided into two groups: (1) linear (1-D) expressions which consist of only $Right$ relationships such as $2+2$, $a+b$; (2) 2-D expressions of which relationships are not only $Right$ relationships such as P^{eo}, $\sqrt{36}$, $\frac{a+b}{c+d}$. There are totally 9817 expressions (8834 for training and 983 for test) in CROHME 2014 data set. Among them, the amount of linear expressions is 2874, accounting for around 30% proportion. Furthermore, we define *chain-SRT expressions* as certain expressions of which the symbol relation trees are essentially a chain structure. *Chain-SRT expressions* contain all the linear expressions and a part of 2-D expressions such as P^{eo}, $\sqrt{36}$. Figure 4.2 illustrates the classifications of expressions.

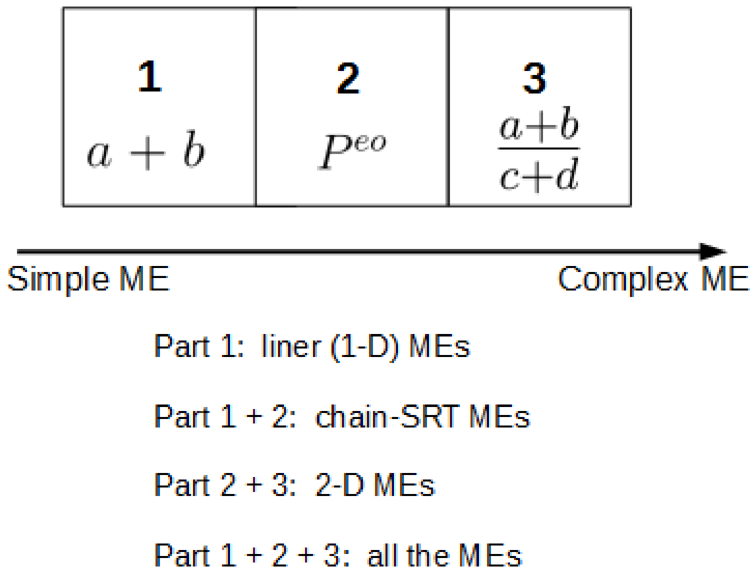
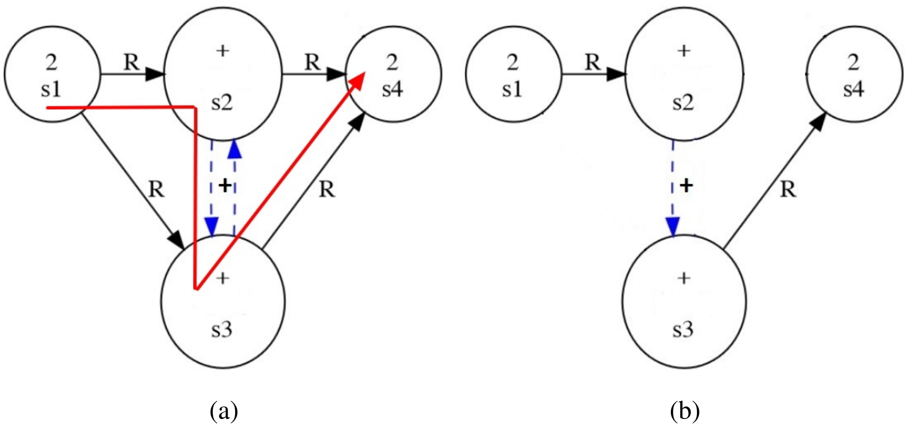


Figure 4.2 – Illustration of the complexity of math expressions.

4.1.2 The proposed idea

Currently in CROHME, SLG is the official format to represent the ground-truth of handwritten math expressions and also for the recognition outputs. The recognition system proposed in this thesis is aiming to output the SLG directly for each input expression. As a strict expression, we use 'correct SLG' to denote the SLG which equals to the ground truth, and 'valid SLG' to represent the graph where double-direction edge corresponds to segmentation information and all strokes (nodes) belonging to one symbol have the same input and output edges. In this section, we explain how to build a valid SLG from a sequence of strokes. An input handwritten mathematical expression consists of one or more strokes. The sequence of strokes in an expression can be described as $S = (s_1, ..., s_n)$. For $i < j$, we assume s_i has been entered before s_j.

A path (*different from the notation within the CTC part*) in SLG can be defined as $\Phi_i = (n_0, n_1, n_2, ..., n_e)$, where n_0 is the starting node and n_e is the end node. The set of nodes of Φ_i is $n(\Phi_i) = \{n_0, n_1, n_2, ..., n_e\}$ and the set of edges of Φ_i is $e(\Phi_i) = \{n_0 \rightarrow n_1, n_1 \rightarrow n_2, ..., n_{e-1} \rightarrow n_e\}$, where $n_i \rightarrow n_{i+1}$ denotes the edge from n_i to n_{i+1}. In fact, the sequence of strokes described as $S = (s_1, ..., s_n)$ is exactly the path following stroke writing order (called $time$ path, Φ_t) in SLG. Still taking '2 + 2' as example, the $time$ path is presented with red color in Figure 4.3a. If all nodes and edges from Φ_t are well classified during the recognition process, we could obtain a chain-SLG as the Fig 4.3b. We propose to get a complete (i.e. valid) SLG from Φ_t by adding the edges which can be deduced from the labeled path to obtain a coherent SLG as depicted on Figure 4.3c. The process can be seen as: (1) complete the segmentation edges between



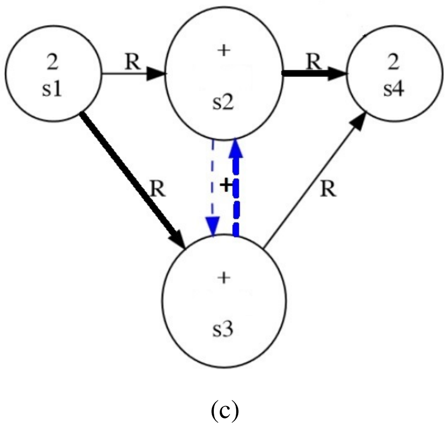


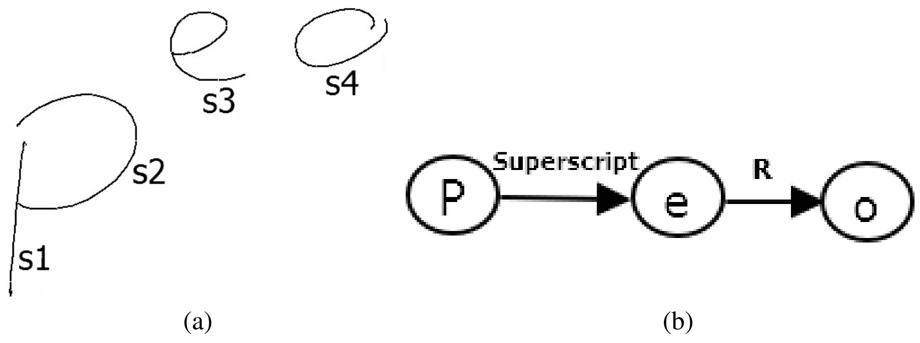
Figure 4.3 – (a) The $time$ path (red) in SLG; (b) the SLG obtained by using the $time$ path; (c) the post-processed SLG of '2 + 2', added edges are depicted as bold.

any pair of strokes of the multi-stroke symbol; (2) add the same input and output relation edges edge for each stroke of the multi-stroke symbol. The $time$ path is used since it is the most intuitive and it is easily available. However, it does not always allow a complete construction of the correct (ground truth) SLG. Different examples are given below to illustrate this point.

Considering both the nodes and edges, we rewrite the $time$ path Φ_t shown in Figure 4.3b as the format of $(s1, s1 \rightarrow s2, s2, s2 \rightarrow s3, s3, s3 \rightarrow s4, s4)$ labeled as $(2, R, +, +, +, R, 2)$. This sequence alternates the node labels $\{2, +, +, 2\}$ and the edge labels $\{R, +, R\}$. Given the labeled sequence $(2, R, +, +, +, R, 2)$, the information that $s2$ and $s3$ belong to the same symbol $+$ can be derived. With the rule that double-direction edge represents segmentation information, the edge from $s3$ to $s2$ will be added automatically. According to the rule that all strokes in a symbol have the same input and output edges, the edges from $s1$

to $s3$ and from $s2$ to $s4$ will be added automatically. The added edges are shown in bold in Figure 4.3c. In this case a correct SLG is built from Φ_t.

Our proposal of building SLG from the $time$ path works well on *chain-SRT expressions* as long as each symbol is written successively and the symbols in such kind of expressions are entered following the order from the root to the leaf in SRT. Successful cases include linear expressions as $2+2$ mentioned previously and a part of 2-D expressions such as P^{eo} shown in Figure 4.4a. The sequence of strokes and edges is $(P, P, P, Superscript, e, R, o)$. All the spatial relationships are covered in it and naturally a correct SLG can be generated. Usually users enter the expression P^{eo} following the order of P, e, o. Yet the input order of e, o, P could be also possible. For this case, the corresponding sequence of strokes and edges is $(e, R, o, \_, P, P, P)$. Since there is no edge from o to P in SLG, we use _ to represent it. Apparently, it is not possible to build a complete and correct SLG with this sequence of labels where the $Superscript$ relationship from P to e is missing. As a conclusion, for a chain-SRT expression written with specific order, a correct SLG could be built using the $time$ path.



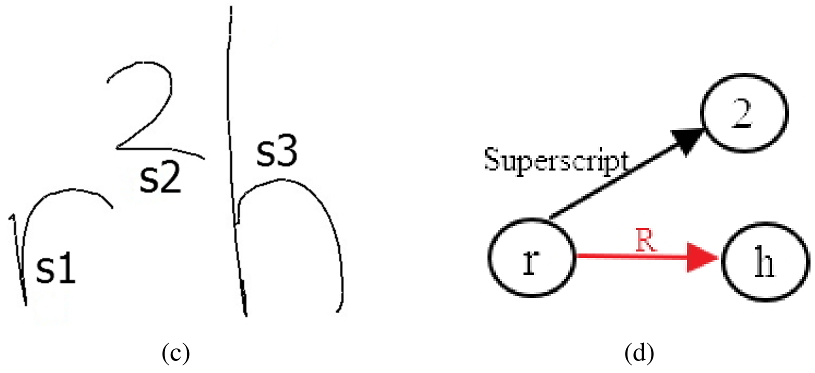


Figure 4.4 – (a) P^{eo} written with four strokes; (b) the SRT of P^{eo}; (c) r^2h written with three strokes; (d) the SRT of r^2h, the red edge cannot be generated by the time sequence of strokes

For those 2-D expressions of which the SRTs are beyond of the chain structure, the proposal presents unbreakable limitations. Figure 4.4c presents a failed case. According to time order, 2 and h are neighbors but there is no edge between them as can be seen on Figure 4.4d. In the best case the system can output a sequence of stroke and edge labels $(r, Superscript, 2, \_, h)$. The $Right$ relationship existing between r and h drawn with red color in Figure 4.4d is missing in the previous sequence. It is not possible to build the correct SLG with $(r, Superscript, 2, \_, h)$. If we change the writing order, first r, h and then 2, the time sequence will be $(r, Right, h, \_, 2)$. Yet, we still can not build a correct SLG with $Superscript$ relationship missing. Being aware of this limitation, the 1-D time sequence of strokes is used to train the BLSTM and the outputted sequence of labels during recognition will be used to generate a valid SLG graph.

4.2 Detailed Implementation

An online mathematical expression is a sequence of strokes described as $S = (s_1, ..., s_n)$. In this section, we present the process to generate the above-mentioned 1-D sequence of labels from S with the

BLSTM and local CTC model. CTC layer only outputs the final sequence of labels while the alignment between the inputs and the labels is unknown. BLSTM with CTC model may emit the labels before, after or during the segments (strokes). Furthermore, it tends to glue together successive labels that frequently co-occur [Graves et al., 2012]. However, the label of each stroke is required to build SLG, which means the alignment information between a sequence of strokes and a sequence of labels should be provided. Thus, we propose local CTC here, constraining the network to emit the label during the segment (stroke), not before or after. First part is to feed the inputs of the BLSTM with S. Then, we focus on the network training process—local CTC methodology. Lastly, the recognition strategies adopted in this chapter will be explained in detail.

4.2.1 BLSTM Inputs

To feed the inputs of the BLSTM, it is important to scan the points belonging to the strokes themselves (on-paper points) as well as the points separating one stroke from the next one (in-air points). We expect that the visible strokes will be labeled with corresponding symbol labels and that the non-visible strokes connecting two visible strokes will be assigned with one of the possible edge labels (could be relationship label, symbol label or '_'). Thus, besides re-sampling points from visible strokes, we also re-sample points from the straight line which links two visible strokes, as can be seen in Figure 4.5. In the rest of this thesis,

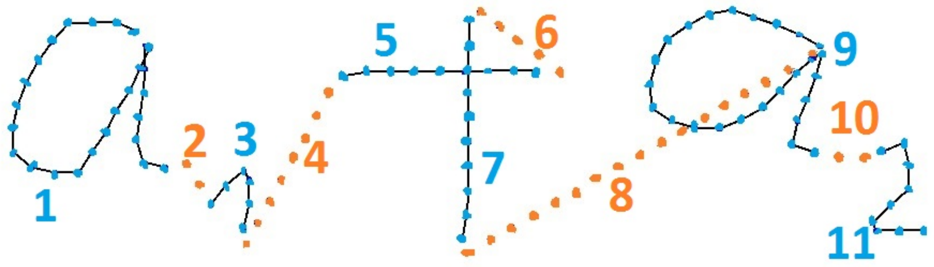


Figure 4.5 – The illustration of on-paper points (blue) and in-air points (red) in $time$ path, $a_1 + a_2$ written with 6 strokes.

$strokeD$ and $strokeU$ are used to indicate a re-sampled pen-down stroke and a re-sampled pen-up stroke for convenience.

Given each expression, we first re-sampled points both from visible strokes and invisible strokes which connects two successive visible strokes in the time order. 1-D unlabeled sequence can be described as $\{strokeD_1, strokeU_2, strokeD_3, strokeU_4, ..., strokeD_K\}$ with K being the number of re-sampled strokes. Note that if s is the number of visible strokes in this path, $K = 2*s-1$. Each stroke ($strokeD$ or $strokeU$) consists of one or more points. At a time-step, the input provided to the BLSTM is the feature vector extracted from one point. Without CTC output layer, the ground-truth of every point is required for BLSTM training process. With CTC layer, only the target labels of the whole sequence is needed, the pre-segmented training data is not required. In this chapter, a local CTC technology is proposed and the ground-truth of each stroke is required. The label of $strokeD_i$ should be assigned with the label of the corresponding node in SLG; the label of $strokeU_i$ should be assigned with the label of the corresponding edge in SLG. If no corresponding edge exists, the label $NoRelation$ will be defined as '_'.

4.2.2 Features

A stroke is a sequence of points sampled from the trajectory of a writing tool between a pen-down and a pen-up at a fixed interval of time. Then an additional re-sampling is performed with a fixed spatial step to get rid of the writing speed. The number of re-sampling points depends on the size of expression. For each expression, we re-sample with $10 \times (length/avrdiagonal)$ points. Here, $length$ refers to the length of all the strokes in the path (including the gap between successive strokes) and $avrdiagonal$ refers to the average diagonal of the bounding boxes of all the strokes in an expression. Since the features used in this work are independent of scale, the operation of re-scaling can be omitted.

Subsequently, we compute five local features per point, which are quite close to the state of art [Álvaro et al., 2013, Awal et al., 2014]. For every point $p_i(x, y)$ we obtained 5 features (see Figure 4.6a):

$$[\sin\theta_i, \cos\theta_i, \sin\phi_i, \cos\phi_i, PenUD_i]$$

with:

- $\sin\theta_i$, $\cos\theta_i$ are the sine and cosine directors of the tangent of the stroke at point $p_i(x, y)$;
- ϕ_i = $\Delta\theta_i$, defines the change of direction at point $p_i(x, y)$;
- $PenUD_i$ refers to the state of pen-down or pen-up.

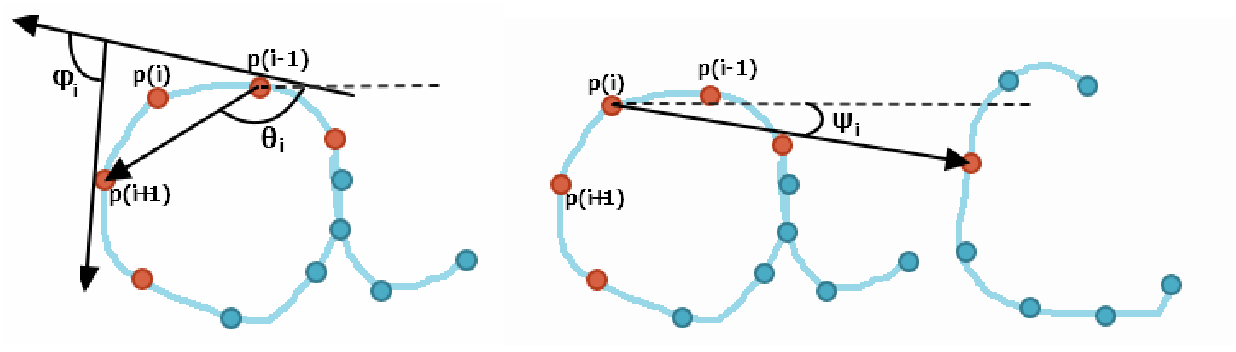


Figure 4.6 – The illustration of (a) θ_i, ϕ_i and (b) ψ_i used in feature description. The points related to feature computation at p_i are depicted in red.

Even though BLSTM can access contextual information from past and future in a long range, it is still interesting to see if a better performance is reachable when contextual features are added in the recognition task. Thus, we extract two contextual features for each point (see Figure 4.6b):

$$[\sin\psi_i, \cos\psi_i]$$

with:

- $\sin\psi_i$, $\cos\psi_i$ are the sine and cosine directors of the vector from the point $p_i(x, y)$ to its closest pen-down point which is not in the current stroke. For the single-stroke expressions, $\sin\psi_i = 0$, $\cos\psi_i = 0$.

Note that the proposed features are size-independent and position-independent characteristics, therefore we omit the normalization process in this thesis. Later in different experiments,we will use the 5 shape descriptor alone or the 7 features together depending on the objective of each experiment.

4.2.3 Training process — local connectionist temporal classification

Frame-wise training of RNNs requires separate training targets for every segment or timestep in the input sequence. Even though presegmented training data is available, it is known that BLSTM and CTC stage have better performance when a 'blank' label is introduced during training [Bluche et al., 2015], so that better decision can be made only at some point in the input sequence. Of course doing so, precise segmentation of the input sequence is not possible. As the label of each stroke is required to build a SLG, we should make decisions on stroke (*strokeD* or *strokeU*) level instead of sequence level (as classical CTC) or point level during the recognition process. Thus, a correspondingly stroke level training method

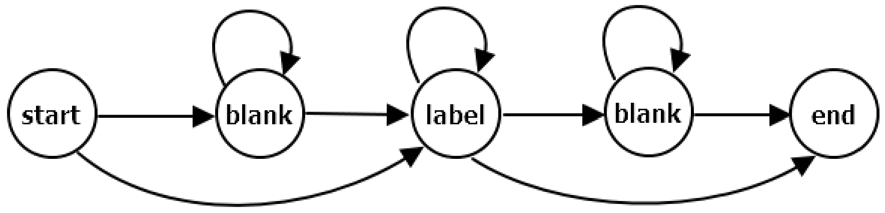


Figure 4.7 – The possible sequences of point labels in one stroke.

allowing the usage of blank label under the constraint of labeling each stroke should be developed. That is why local CTC is proposed here.

For each stroke, label sequences should follow the state diagram given in Figure 4.7. 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.

In Section 3.4, CTC technology proposed by Graves is introduced. We modify the CTC algorithm with a local strategy to let it output the relatively precise alignment between the input sequence and the output sequence of labels. In this way, it could be applied for the training stage in our proposed system. Given the input sequence X of length T consisting of U strokes, l is used to denote the ground truth, i.e. the sequence of labels. As one stroke belongs to at most one symbol or one relationship, the length of l is U. l' represents the label sequence with blanks added to the beginning and the end of l, and inserted between every pair of consecutive labels. Apparently, the length of l' is $U' = 2U + 1$. The forward variable $\alpha(t, u)$ denotes the summed probability of all length t paths that are mapped by F onto the length $u/2$ prefix of l, where u is from 1 to U' and t is from 1 to T. Given the above notations, the probability of l can be expressed as the sum of the forward variables with and without the final blank at time T.

$$p(l|X) = \alpha(T, U') + \alpha(T, U' - 1) \tag{4.1}$$

In our case, $\alpha(t, u)$ can be computed recursively as following:

$$\alpha(1, 1) = y^1_{-} \tag{4.2}$$

$$\alpha(1, 2) = y^1_{l_1} \tag{4.3}$$

$$\alpha(1, u) = 0, \forall u > 2 \tag{4.4}$$

$$\alpha(t, u) = y^t_{l'_u} \sum_{i=f_{local}(u)}^{u} \alpha(t-1, i) \tag{4.5}$$

where

$$f_{local}(u) = \begin{cases} u-1 & \text{if } l'_u = \text{blank} \\ u-2 & \text{otherwise} \end{cases} \tag{4.6}$$

In the original Eqn. 3.34, the value $u - 1$ was also assigned when $l'_{u-2} = l'_u$, enabling the transition from $\alpha(t-1, u-2)$ to $\alpha(t, u)$. This is the case when there are two repeated successive symbols in the final labeling. With regard to the corresponding paths, there exists at least one blank between these two symbols. Otherwise, only one of these two symbols can be obtained in the final labeling. In our case, as one label will be selected for each stroke, the above-mentioned limitation can be ignored. Suppose that the input at time t belongs to i^{th} stroke (i from 1 to U), then we have

$$\alpha(t,u) = 0, \forall u/u < (2*i-1), u > (2*i+1) \tag{4.7}$$

which means the only possible arrival positions for time t are l'_{2*i-1}, l'_{2*i}, l'_{2*i+1}. Figure 4.8 demonstrates the local CTC forward-backward algorithm using the example '2a' which is written with 2 visible strokes. The

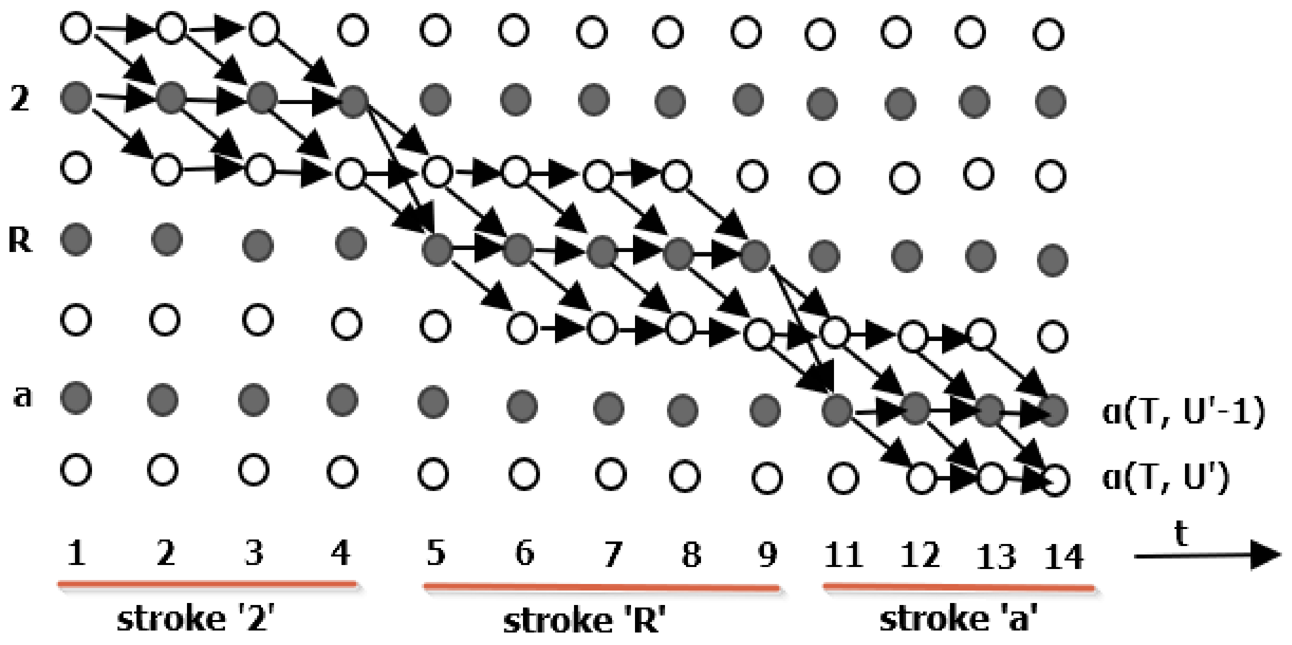


Figure 4.8 – Local CTC forward-backward algorithm. Black circles represent labels and white circles represent blanks. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated in the reverse direction.

corresponding label sequences l and l' of it are '2Ra' and '-2-R-a-' respectively (R is for $Right$ relationship). We re-sampled 4 points for pen-down stroke '2', 5 points for pen-up stroke 'R' and 4 points for pen-down stroke 'a'. From this figure, we can see each part located on one stroke is exactly the CTC forward-backward algorithm. That is why the output layer adopted in this paper is called local CTC.

Similarly, the backward variable $\beta(t,u)$ denotes the summed probabilities of all paths starting at $t+1$ that complete l when appended to any path contributing to $\alpha(t,u)$. The formulas for the initialization and recursion of the backward variable in local CTC are as follows:

$$\beta(T,U') = 1 \tag{4.8}$$

$$\beta(T,U'-1) = 1 \tag{4.9}$$

$$\beta(T,u) = 0, \forall u < U'-1 \tag{4.10}$$

$$\beta(t,u) = \sum_{i=u}^{g_{local}(u)} \beta(t+1,i) y^{t+1}_{l'_i} \tag{4.11}$$

where

$$g_{local}(u) = \begin{cases} u+1 & \text{if } l'_u = \text{blank} \\ u+2 & \text{otherwise} \end{cases} \tag{4.12}$$

Suppose that the input at time t belongs to i^{th} stroke (i from 1 to U), then:

$$\beta(t,u) = 0, \forall u/u < (2*i-1), u > (2*i+1) \tag{4.13}$$

With the local CTC forward-backward algorithm, the $\alpha(t,u)$ and $\beta(t,u)$ are available for each time step t and each allowed positions u of time step t. Then the errors are backpropagated to the output layer (Equation 3.49), the hidden layer (Equation 3.50), finally to the entire network. The weights in the network are adjusted with the expectation to enabling the network output the corresponding label for each stroke.

As can be seen in Figure 4.8, each part located on one stroke is exactly the CTC forward-backward algorithm. In this chapter, a sequence consisting U strokes is regarded and processed as a entirety. In fact, each stroke i could be coped with separately. To be specific, with regard to each stroke i we have $\alpha_i(t,u)$, $\beta_i(t,u)$ and $p(l_i|X_i)$ associated to it. The initialization of $\alpha_i(t,u)$ and $\beta_i(t,u)$ is the same as described previously. With this treatment, $p(l|X)$ can be expressed as:

$$p(l|X) = \prod_{i=1}^{U} p(l_i|X_i) \tag{4.14}$$

Either way, the result is the same. We will reintroduce this point in Chapter 6 where the separate processing method is taken.

4.2.4 Recognition Strategies

Once the network is trained, we would ideally label some unknown input sequence X by choosing the most probable labeling I^*:

$$I^* = \underset{l}{argmax}\ p(l|X) \tag{4.15}$$

Since local CTC is already adopted in the training process in this work, naturally recognition should be performed at stroke ($strokeD$ and $strokeU$) level. As explained in Section 4.1 to build the Label Graph, we need to assign one single label to each stroke. At that stage, for each point or time step, the network outputs the probabilities of this point belonging to different classes. Hence, a pooling strategy is required to go from the point level to the stroke level. We propose two kinds of decoding methods: maximum decoding and local CTC decoding, both based on stroke level.

Maximum decoding With the same method taken in [Graves et al., 2012] for isolated handwritten digits recognition using a multidimensional RNN with LSTM hidden layers, we first calculate the cumulative probabilities over the entire stroke. For stroke i, let $o^i = \{p^i_{ct}\}$, where p^i_{ct} is the probability of outputting the c^{th} label at the t^{th} point. Suppose that we have N classes of labels (including blank), then c is from 1 to N; $|s_i|$ points are re-sampled for stroke i, then t is from 1 to $|s_i|$. Thus, the cumulative probability of outputting the c^{th} label for stroke i can be computed as

$$P^i_c = \sum_{t=1}^{|s_i|} p^i_{ct} \tag{4.16}$$

Then we choose for stroke i the label with the highest P^i_c (excluding $blank$).

Local CTC decoding With the output o^i, we choose the most probable label for the stroke i:

$$l^*_i = \underset{l_i}{argmax}\ p(l_i|o^i) \tag{4.17}$$

In this work, each stroke outputs only one label which means we have $N-1$ possibilities of label of stroke. $blank$ is excluded because it can not be a candidate label for stroke. With the already known $N-1$ labels, $p(l_i|o^i)$ can be calculated using the algorithm depicted in Section 4.2.3. Specifically, based on the Eqn. 6.17 we can write Eqn. 4.18,

$$p(l_i|o^i) = \alpha(|s_i|, 3) + \alpha(|s_i|, 2) \tag{4.18}$$

with $T = |s_i|$ and $U' = 3$ (l' is $(blank, label, blank)$). For each stroke, we compute the probabilities corresponding to $N-1$ labels and then select the one with the largest value. In mathematical expression recognition task, more than 100 different labels are included. If Eqn. 4.18 is computed more that 100 times for every stroke, undoubtedly it would be a time-consuming task. A simplified strategy is adopted here. We sort the P_c^i from Eqn. 4.16 using maximum decoding and keep the top 10 probable labels (excluding $blank$). From these 10 candidates, we choose the one which has the highest $p(l_i|o^i)$. In this way, Eqn. 4.18 is computed only 10 times for each stroke, greatly reducing the computation time.

Furthermore, we add two constraints when choosing label for stroke: (1) the label of $strokeD$ should be one of the symbol labels, excluding the relationship labels, like strokes $1, 3, 5, 7, 9, 11$ in Figure 4.9. (2) the label of $strokeU_i$ is divided into 2 cases, if the labels of $strokeD_{i-1}$ and $strokeD_{i+1}$ are different, it should be one of the six relationships (strokes $2, 8, 10$) or '_' (stroke 4); otherwise, it should be relationships, '_' or the label of $strokeD_{i-1}$ ($strokeD_{i+1}$). Taking stroke 6 shown in Figure 4.9 for example, if '$+$' is assigned to it means that the corresponding pair of nodes (strokes 5 and 7) belongs to the same symbol while '_' or relationship refers to 2 nodes belonging to 2 symbols. Note that to satisfy these constraints on edges labels, the labels of pen-down strokes are chosen first and then pen-up strokes.

After recognition, post-processing (adding edges) should be done in order to build the SLG. The way to proceed has been already introduced in Section 4.1.

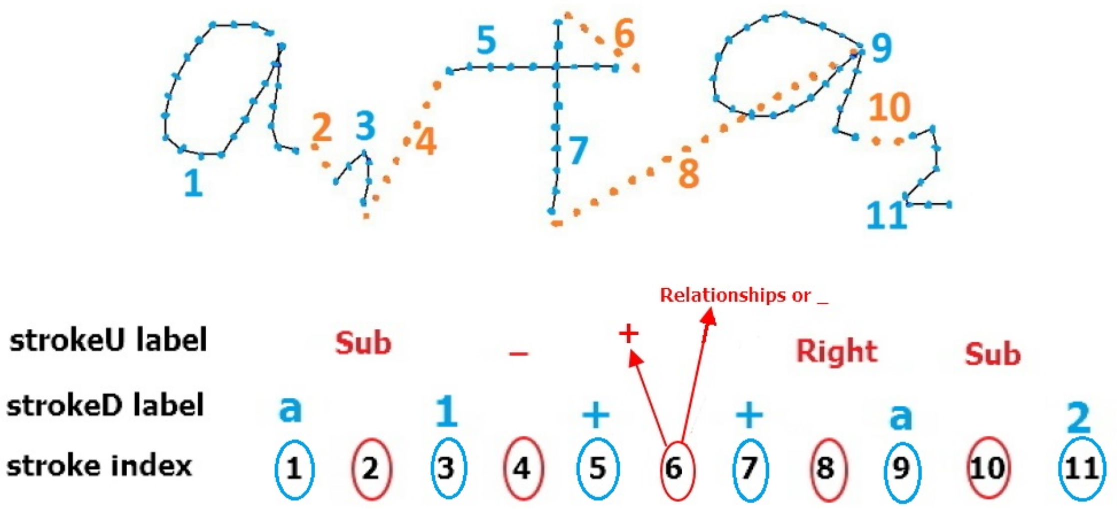


Figure 4.9 – Illustration for the decision of the label of strokes. As stroke 5 and 7 have the same label, the label of stroke 6 could be '$+$', '_' or one of the six relationships. All the other strokes are provided with the ground truth labels in this example.

4.3 Experiments

We extend the RNNLIB library [1] by introducing the local CTC training technique, and use the extended library to train several BLSTM models. Both frame-wise training and local CTC training are adopted in our experiments. For each training process, the network having the best classification error (frame-wise) or

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

CTC error (local CTC) on validation data set is saved. Then, we test this network on the test data set. The maximum decoding (Eqn. 4.16) is used for frame-wise training network. With regard to local CTC, either the maximum decoding or local CTC decoding (Eqn. 4.18) can be used.

With the Label Graph Evaluation library (LgEval) [Mouchère et al., 2014], 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; 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 right relationship label.

For all experiments the network architecture and configuration are as follows:

- The input layer size: 5 or 7 (when considering the 2 additionnal context features)
- The output layer size: the number of class (up to 109)
- The hidden layers: 2 layers, the forward and backward, each contains 100 single-cell LSTM memory blocks
- The weights: initialized uniformly in [-0.1, 0.1]
- The momentum: 0.9

This configuration has obtained good results in both handwritten text recognition [Graves et al., 2009] and handwritten math symbol classification [Álvaro et al., 2013, 2014a].

4.3.1 Data sets

Being aware of the limitations of our proposal related to the structures of expressions, we would like to see the performance of the current system on expressions of different complexities. Thus, three data sets are considered in this chapter.

Data set 1. We select the expressions which do not include 2-D spatial relation, only left-right relation from CROHME 2014 training and test data. 2609 expressions are available for training, about one third of the full training set and 265 expressions for testing. In this case, there are 91 classes of symbols. Next, we split the training set into a new training set and validation set, 90% for training and 10% for validation. The output layer size is 94 (91 symbol classes + $Right$ + $NoRelation$ + $blank$). In left-right expressions, $NoRelation$ will be used each time when a delayed stroke breaks the left-right time order.

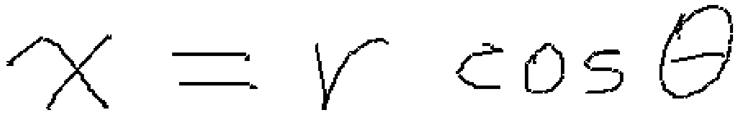
Data set 2. The depth of expressions in this data set is limited to 1, which imposes that two sub-expressions having a spatial relationship ($Above$, $Below$, $Inside$, $Superscript$, $Subscript$) should be left-right expressions. It adds to the previous linear expressions some more complex MEs. 5820 expressions are selected for training from CROHME 2014 training set; 674 expressions for test from CROHME 2014 test set. Also, we divide 5820 expressions into the new training set and validation set, 90% for training and 10% for validation. The output layer size is 102 (94 symbol classes + 6 relationships + $NoRelation$ + $blank$).

Data set 3. The complete data set from CROHME 2014, 8834 expressions for training and 983 expressions for test. Also, we divide 8834 expressions for training (90%) and validation (10%). The output layer size is 109 (101 symbol classes + 6 relationships + $NoRelation$ + $blank$).

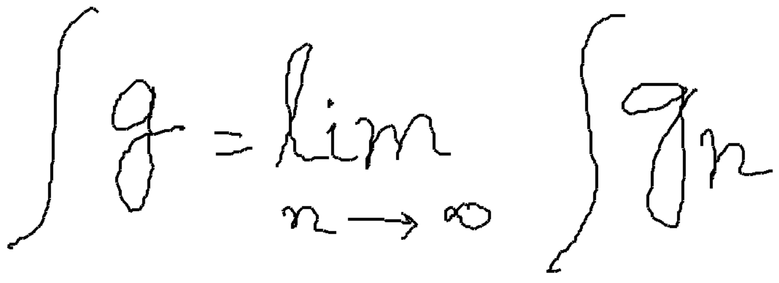
The $blank$ label is only used for local CTC training. Figure 4.10 show some handwritten math expression samples extracted from CROHME 2014 data set.

4.3.2 Experiment 1: theoretical evaluation

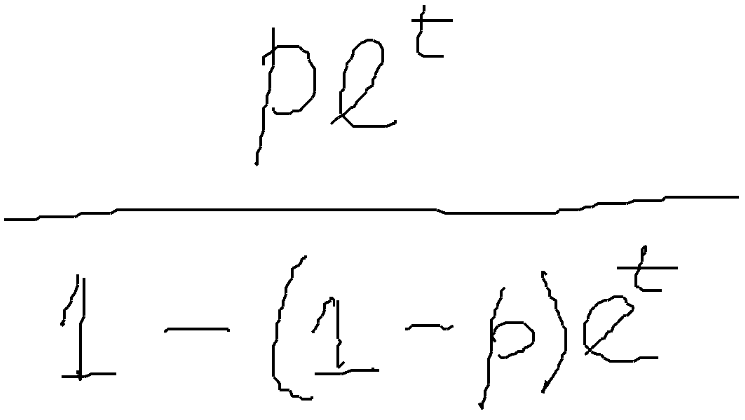
As discussed in Section 4.1, there exist obvious limitations in the proposed solution of this chapter. These limitations could be divided into two types: (1) to *chain-SRT expressions*, if users could not write a multi-stroke symbol successively or could not follow a specific order to enter symbols, it will not be possible to build a correct SLG; (2) to those expressions of which the SRTs are beyond of the chain structure, regardless of the writing order, the proposed solution will miss some relationships. In this experiment,



(a)



(b)



(c)

Figure 4.10 – Real examples from CROHME 2014 data set. (a) sample from **Data set 1**; (b) sample from **Data set 2**; (c) sample from **Data set 3**.

laying the classifier aside temporarily, we would like to evaluate the limitations of the proposal itself. Thus, to carry out this theoretical evaluation, we take the ground truth labels of the nodes and edges in the $time$ path only of each expression. Table 4.1 and Table 4.2 present the evaluation results on CROHME 2014 test set at the symbol and expression level respectively using the above-mentioned strategy.

We can see from Table 4.1, the recall ('Rec.') and precision ('Prec.') rates of the symbol segmentation on all these 3 data sets are almost 100% which implies that users generally write a multi-stroke symbol successively. The recall rate of the relationship recognition is decreasing from **Data set 1** to **3** while the precision rate remains almost 100%. With the growing complexity of expressions, increasing relationships are missed due to the limitations. About 5% relationships are missed in **Data set 1** because of only the problem of writing order. With regards to the approximate 25% relationships omitted in **Data set 3**, it is owing to the writing order and the conflicts between the chain representation method and the tree structure of expression, especially the latter one.

In Table 4.2, the evaluation results at the expression level are available. 86.79% of **Data set 1** which contains only 1-D expressions could be recognized correctly with the proposal at most. For the complete CROHME 2014 test set, only 34.11% expressions can be interpreted correctly in the best case.

Table 4.1 – The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels on the $time$ path).

Data set	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
1	99.73	99.46	99.73	99.46	95.78	99.40
2	99.75	99.49	99.73	99.48	80.33	99.39
3	99.73	99.45	99.72	99.44	75.54	99.27

Table 4.2 – The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels on the $time$ path).

Data set	correct (%)	<= 1 error	<= 2 errors	<= 3 errors
1	86.79	87.55	91.32	93.96
2	44.21	51.63	61.87	68.69
3	34.11	40.94	50.51	58.25

4.3.3 Experiment 2

In this experiment, we evaluate the proposed solution with BLSTM classifier on data sets of different complexity. Local CTC training and local CTC decoding methods are used inside the recognition system. Only 5 local features are extracted at each point for training. Each system is trained only once.

The evaluation results on symbol level for the 3 data sets are provided in Table 4.3 including recall ('Rec.') and precision ('Prec.') rates for 'Segments', 'Seg+Class', 'Tree Rels.'. As can be seen, the results in 'Segments' and 'Seg+Class' are increasing while the training data set is growing. The recall for 'Tree Rels.' is decreasing among the three data sets. It is understandable since the number of missed relationships grows with the complexity of expressions knowing the limitation of our method. The precision for 'Tree Rels.' fluctuates as the data set is expanding. The results of **Data set 3** are comparable to the results of CROHME 2014 because the same training and testing data sets are used. The second part of Table 4.3 gives the symbol level evaluation results of the participant systems in CROHME 2014 sorted by recall of correct symbol segmentation. The best 'Rec.' of 'Segments' and 'Seg+Class' reported in CROHME 2014 are 98.42% and 93.91% respectively. Ours are 93.26% and 84.40%, both ranked 3 out of 8 systems (7 participants in CROHME 2014 + our system). Our solution presents competitive results on symbol recognition task and segmentation task even though the symbols with delayed strokes are missed. However, our proposal, at that stage, shows limited performances at the relationship level, with 'Rec.' = 61.85%, 'Prec.' = 75.06%. This is mainly because approximate 25% relationships are missed in the time sequence. If we consider only the relationships covered by the time sequence which accounts for 75.54%, the recall rate will be 61.85%/75.54% = 81.88%, close to the second ranked system in the competition. Thus, one of the main works of next chapters would be focused on proposing a solution to catch the omitted approximate 25% relationships at the modeling stage.

Table 4.3 – The symbol level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results.

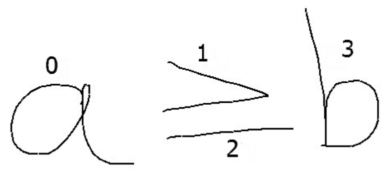
Data set, features	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
1, 5	90.11	80.75	78.91	70.71	79.87	73.66
2, 5	91.88	84.47	82.42	75.77	64.75	71.96
3, 5	**93.26**	**86.86**	**84.40**	**78.61**	**61.85**	**75.06**
system	CROHME 2014 participant results					
III	98.42	98.13	93.91	93.63	94.26	94.01
I	93.31	90.72	86.59	84.18	84.23	81.96
VII	89.43	86.13	76.53	73.71	71.77	71.65
V	88.23	84.20	78.45	74.87	61.38	72.70
IV	85.52	86.09	76.64	77.15	70.78	71.51
VI	83.05	85.36	69.72	71.66	66.83	74.81
II	76.63	80.28	66.97	70.16	60.31	63.74

Table 4.4 shows the recognition rates at the global expression level with no error, and with at most one to three errors in the labels of SLG. This metric is very strict. For example one label error can happen only on one stroke symbol or in the relationship between two one-stroke symbols; a labeling error on a 2-stroke symbol leads to 4 errors (2 nodes labels and 2 edges labels). As can be seen, the expression recognition rates are decreasing as the data sets are getting more and more complex from **Data set 1** to **3**. On **Data set 1** of only linear expressions, the ME recognition rate is 25.28%. The recognition rate with no error on CROHME 2014 test set is 12.63%. The best one and worst one reported by CROHME 2014 are 62.68% and 15.01%. When looking at the recognition rate having less than three errors, four participants ranked between 27% and 37%, while our result is 31.98%.

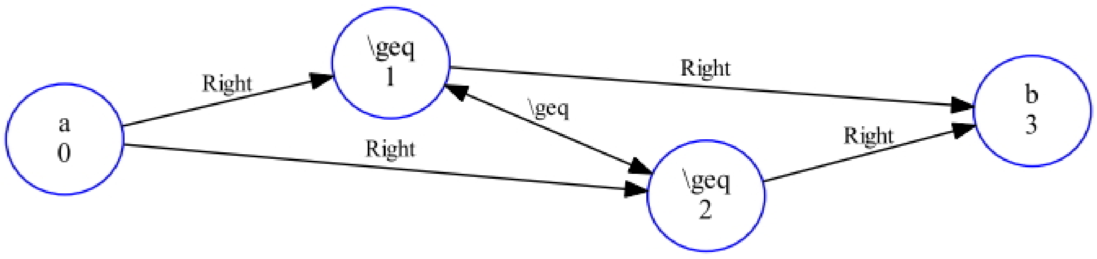
Table 4.4 – The expression level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results.

Data set, features	correct (%)	<= 1 error	<= 2 errors	<= 3 errors
1, 5	25.28	40.75	49.06	52.08
2, 5	12.76	25.07	31.16	36.20
3, 5	**12.63**	**21.28**	**27.70**	**31.98**
system	CROHME 2014 participant results			
III	62.68	72.31	75.15	76.88
I	37.22	44.22	47.26	50.20
VII	26.06	33.87	38.54	39.96
VI	25.66	33.16	35.90	37.32
IV	18.97	28.19	32.35	33.37
V	18.97	26.37	30.83	32.96
II	15.01	22.31	26.57	27.69

We present a correctly recognized sample and an incorrectly recognized sample in Figure 4.11 and Figure 4.12 respectively. The expression $a \geq b$ (Figure 4.11) is a 1-D expression and therefore the $time$

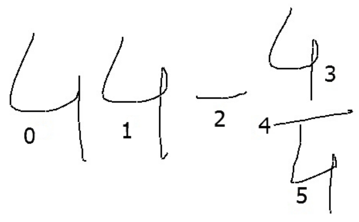


(a)

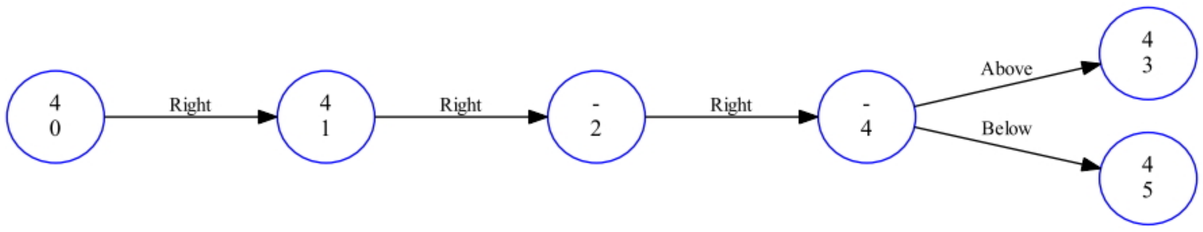


(b)

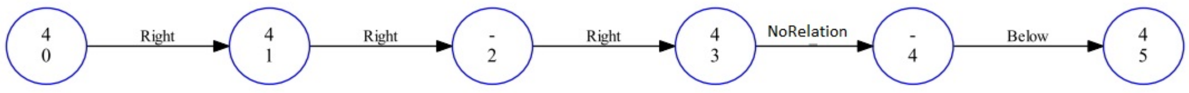
Figure 4.11 – (a) $a \geq b$ written with four strokes; (b) the built SLG of $a \geq b$ according to the recognition result, all labels are correct.



(a)



(b)



(c)

Figure 4.12 – (a) $44 - \frac{4}{4}$ written with six strokes; (b) the ground-truth SLG; (c) the rebuilt SLG according to the recognition result. Three edge errors occurred: the $Right$ relation between stroke 2 and 4 was missed because there is no edge from stroke 2 to 4 in the time path; the edge from stroke 4 to 3 was missed for the same reason; the edge from stroke 2 to 3 was wrongly recognized and it should be labeled as $NoRelation$.

path could cover all the relationships in this expression. It was correctly recognized by our system in this chapter. Considering the other sample $44 - \frac{4}{4}$ of which the SRT is a tree structure (Figure 4.12), the $Right$ relationship from the minus symbol to fraction bar was omitted in the modeling stage, likewise the $Above$

relationship from the fraction bar to the numerator 4. In addition, the relation from the minus symbol to the numerator 4 was wrongly recognized as $Right$ and it should be labeled as $NoRelation$.

4.3.4 Experiment 3

In this experiment, we would like to know if different training and decoding methods and the contextual features will improve or not the performance of our recognition system. We use different training methods and different features to train the recognition system, and take two kinds of strategy at the recognition stage. All the systems in this part are trained and evaluated on **Data set 3**. Since the weights inside the network are initialized randomly, each system is trained four times with the aim to compute mean evaluation values and standard deviations, and therefore obtain convincing conclusions and have an idea of the system stability.

As shown in Table 4.5, with the first 2 networks, we can conclude that local CTC training can improve

Table 4.5 – The symbol level evaluation results (mean values) on CROHME 2014 test set with different training and decoding methods, features.

Feat.	Train	Decode	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
			Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
5	frame-wise	maximum	92.71	85.88	83.76	77.59	59.84	73.71
5	local CTC	maximum	93.21	86.73	84.11	78.26	61.75	74.51
5	local CTC	local CTC	93.2	86.71	84.11	78.25	61.71	74.67
7	local CTC	maximum	93	86.43	84	78.06	61.73	74.06

the system performance globally compared to frame-wise training. Furthermore, the proposed local CTC training method is able to accelerate the convergence process and therefore reduce the training time significantly. Comparing the results from the second and third systems, the conclusion is straightforward that local CTC decoding does not help the recognition process but cost more computation. Maximum decoding is a better choice in this work. In the fourth system, we test the effect of contextual features on BLSTM networks. The results stay at the same level with system trained with only local features. In addition, to have a look at the stability of our recognition system, we provide in Table 4.6 the standard derivations of the symbol level evaluation results on CROHME 2014 test set with local CTC training and maximum decoding methods, 5 local features. As shown, the standard derivations are quite low, indicating that our system has a high stability.

Table 4.6 – The standard derivations of the symbol level evaluation results on CROHME 2014 test set with local CTC training and maximum decoding method, 5 local features.

Feat.	Train	Decode	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
			Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
5	local CTC	maximum	0.12	0.26	0.26	0.32	0.29	0.83

Consequently, in the coming chapters we will use **local CTC training** method instead of frame-wise training, **maximum decoding** instead of local CTC decoding, **5 local features** instead of 7 features for all the experiments aimed at making an effective and efficient system.

4.4 Discussion

The capability of BLSTM networks to process graphical two-dimensional languages such as handwritten mathematical expressions is explored in this chapter as a first try. Using online math expressions, which are available as a temporal sequence of strokes, we produce a labeling at the stroke level using a BLSTM network with a local CTC output layer. Then we propose to build a two-dimensional (2-D) expression from this sequence of labels. Our solution presents competitive results with CROHME 2014 data set on symbol recognition task and segmentation task. Proposing a global solution to perform at one time segmentation, recognition and interpretation, with no dedicated stages, is a major advantage of the proposed solution. To some extent, at the present time, it fails on the relationship recognition task. This is primarily due to an intrinsic limitation, since currently, a single path following the time sequence of strokes in SLG is used to build the expression. In fact, some important relationships are omitted at the modeling stage.

We only considered stroke combinations in time series in the work of this chapter. For the coming chapter, the proposed solution will take into account more possible stroke combinations in both time and space such that less relationships will be missed at the modeling stage. A sequential model could not include temporal and spacial information at the same time. To overcome this limitation, we propose to build a graph from the time sequence of strokes to model more accurately the relationships between strokes.

5

Mathematical expression recognition by merging multiple paths

In Chapter 4, we confirmed the fact of that there exists unbreakable limitations if using a single 1-D path to model expressions. This conclusion was verified from both theoretical and experimental point of view. The sequence of strokes arranged with time order was used in those experiments as an example of 1-D paths since it is the most intuitive and readily available. Due to the unbreakable limitations, in this chapter, we turn to a graph structure to model the relationships between strokes in mathematical expressions. Further, using the sequence classifier BLSTM to label the graph structure is another research focus.

This chapter will be organized as follows: Section 5.1 provides an overview of graph representation related to build a graph from raw mathematical expression. Then we globally describe the framework of mathematical expression recognition by merging multiple paths in Section 5.2. Next, all the steps of the recognition system are explained one by one in detail. Finally, the experiment part and the discussion part are presented in Section 5.4 and Section 5.5 respectively.

5.1 Overview of graph representation

Each mathematical expression consists of a sequence of strokes. Relations between two strokes could be divided into 3 types: *belong to the same symbol (segmentation)*, *one of the 6 spatial relationships*, *no relation*. It is possible to describe a ME at the stroke level using a SLG of which nodes represent strokes, while the edges encode either segmentation information or one of the spatial relationships. If there is *no relation* between two strokes, of course no corresponding edge would be found between two strokes in SLG.

All the above discussion supposes the knowledge of the ground truth. In fact, given a handwritten expression, our work is to find the ground truth. Thus the first step is to derive an intermediate graph from the raw information. Specifically, it involves finding pairs of strokes between which there exist relations (represented as edges). In [Hu, 2016], they call this stage as **graph representation**. We could find out all the ground truth edges (100% recall) by adding an edge between any pair of strokes in the derived graph. However, this exhaustive approach brings at the same time the problem of low precision. For an expression with N strokes, if we consider all the possibilities, there would be $N(N-1)$ edges in the derived graph. Compared to the ground truth SLG, many edges do not exist. Suppose that all symbols in this expression are single-stroke symbol, there are only $N-1$ ground truth edge, and the precision is $\frac{1}{N}$. Apparently this exhaustive solution with a 100% recall and an around $\frac{1}{N}$ precision is unbearable in practice because even through later classifier could recognize these invalid edges to some extent it is still a big burden. Thus, better graph models should be explored. In this section, we introduce several models used in other literature provided as a basis of the proposed model in this thesis.

Time Series (TS) is widely used as a model for the task of math symbol segmentation and recognition in previous works [Hu and Zanibbi, 2013, Koschinski et al., 1995, Kosmala and Rigoll, 1998, Yu et al., 2007, Smithies et al., 1999, Winkler and Lang, 1997a,b]. In this model, strokes are represented as nodes, and between two successive strokes in the input order there is an edge (undirected) connecting them. We also considered this model but a directed version in Chapter 4 where it is called $time$ path. Time Series is a good model for symbol segmentation and recognition since people usually writes symbols with no delayed stroke. However, it is not strong enough to capture the global structure of math expression, which has been clarified very well in last chapter.

Unlike **Time Series** which is a chain structure in fact, **K Nearest Neighbor (KNN)** is a graph model in the true sense. In KNN graph, for each stroke, we first search for its K closest strokes. Then each undirected edge between this stroke and each of its K closest neighbors will be added into the graph. Thus, each node has at least K edges connected to it. In other words, the number of the edges connected to each node is relatively fixed in fact. However, it is not well suitable for math expression where nodes are connected to a variable number of edges.

In [Matsakis, 1999], **Minimum spanning tree (MST)** is used as the graph model. A spanning tree is a connected undirected graph, which a set of edges connect all of the nodes of the graph with no cycles. To define a minimum spanning tree, the graph edges also need to be assigned with a weight, in which case an MST is a spanning tree that has the minimum accumulative edge weight of the graph [Matsakis, 1999]. Minimum spanning trees can be efficiently computed using the algorithms of Kruskal and Prim [Cormen, 2009]. In [Matsakis, 1999], each stroke is represented as a node and the edge between the two strokes is assigned with a weight which is the distance of two strokes. Figure 5.1a presents an example of MST.

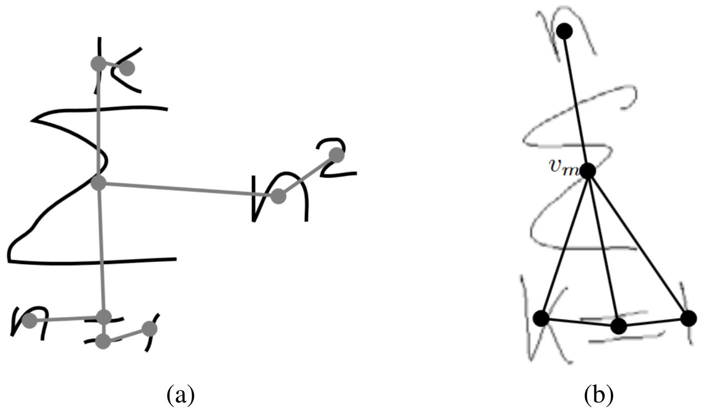


Figure 5.1 – Examples of graph models. (a) An example of minimum spanning tree at stroke level. Extracted from [Matsakis, 1999]. (b) An example of Delaunay-triangulation-based graph at symbol level. Extracted from [Hirata and Honda, 2011].

A **Delaunay Triangulation (DT)** for a set P of points in a plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$ [de Berg et al.]. [Hirata and Honda, 2011] uses a graph model based on Delaunay triangulation. They assume that symbols are correctly segmented, thus, instead of stroke, each symbol is taken as a node. Figure 5.1b presents an example of Delaunay-triangulation-based graph applied on math expressions.

In [Hu, 2016], **Line Of Sight (LOS)** graph is considered since they find that, for a given stroke, it usually can see the strokes which have a relation with it in the symbol relation tree. The center of each stroke is taken as an eye, there would be an directed edge from the current stroke to each stroke it can see. A sample is available in Figure 5.2. [Muñoz, 2015] use an undirected graph of which each stroke is a node and edges only connect strokes that are visible and close as a segmentation model. They also consider the visibility between strokes.

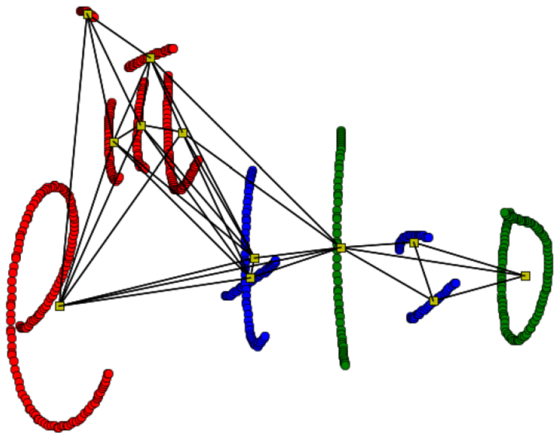
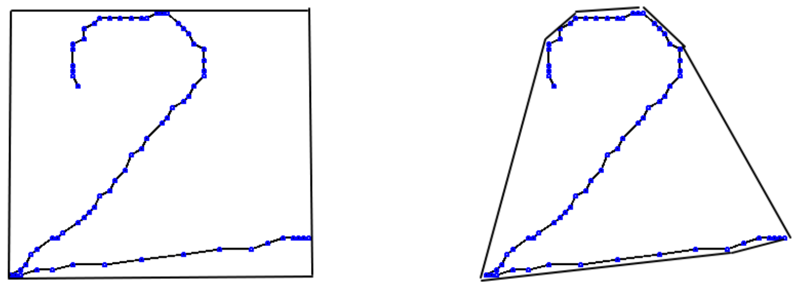


Figure 5.2 – An example of line of sight graph for a math expression. Extracted from [Hu, 2016].

Hu carried out considerable experiments to choose the appropriate graph model in [Hu, 2016]. To better stress the characteristics of variable graph models, several related definitions are provided before going to the details. A stroke is a sequence of points which can be represented as a Bounding Box (BB) (Figure 5.3a) or a Convex Hull (CH) (Figure 5.3b). A subset P of the plane is called convex if and only if for any pair of



(a) (b)

Figure 5.3 – Stroke representation. (a) The bounding box. (b) The convex hull.

points $p_1, p_2 \in P$ the line segment $\overline{p_1p_2}$ is completely contained in P. The convex hull $CH(P)$ of a set P is the smallest convex set that contains P [de Berg et al.]. Three different euclidean distances are proposed to compute the distance between two strokes: (1) the distance between averaged centers. Averaged Center (AC) of a stroke is defined by the average of the x-coordinates and the average of the y-coordinates of all points belonging to this stroke. (2) the distance between Bounding Box Center (BBC) of a pair of strokes. (3) the distance between their closest points. Closest Point Pair (CPP) of two strokes refers to the pair of points having the minimal distance, where two points are from two strokes. The experiment results from [Hu, 2016] reveal that both MST and TS models achieve a high precision rate (around 90% on CROHME 2014 test set, AC distance is used for MST) but a relatively low recall (around 87% on CROHME 2014 test set). For KNN graph, the larger K is, the higher the recall is and the lower the precision is. When $K = 6$, the recall reach 99.4% and the precision is 28.3% (CROHME 2014 test set, CPP distance is used). The recall and the precision of DT graph are 97.3% and 39.1% respectively (CROHME 2014 test set, AC distance is used). In LOS graph model, the bounding box center is taken as eye of a stroke. Each stroke s_i has an Unblocked Angle Range (UAR) which is initialized as $[0, 2\pi]$. For any other stroke s_j, the Block

Angle Range (BAR) of the convex hull is calculated. If the Visibility Angle Range (VAR) = overlap of BAR and UAR is nonzero, s_i could see s_j. Then UAR is updated with UAR - VAR. The model called LOS CH symmetric (CH: the convex hull is used to compute the block angle range of each stroke; symmetric: for a edge from s_i to s_j, there would be the reverse edge from s_j to s_i) has a recall 99.9% of and a precision of 29.7% (CROHME 2014 test set, CPP distance is used). In fact, each graph model has its strong points and also limitations. Hu choose LOS CH symmetric as the graph representation in his work as a high recall and a reasonable precision is required.

Based on the previous works and our own work in Chapter 4, we develop a new graph representation model which is a directed graph built using both the temporal and spatial information of strokes.

5.2 The framework

In this section, we introduce the global framework (Figure 5.4) of the proposed solution of this chapter to provide the readers an intuitive look at the detailed implementation is proposed next. As depicted in Figure 5.4, the input to the recognition system is an handwritten expression which is a sequence of strokes; the output is the stroke label graph which consists of the information about the label of each stroke and the relationships between stroke pairs. As the first step, we derive an intermediate graph from the raw input

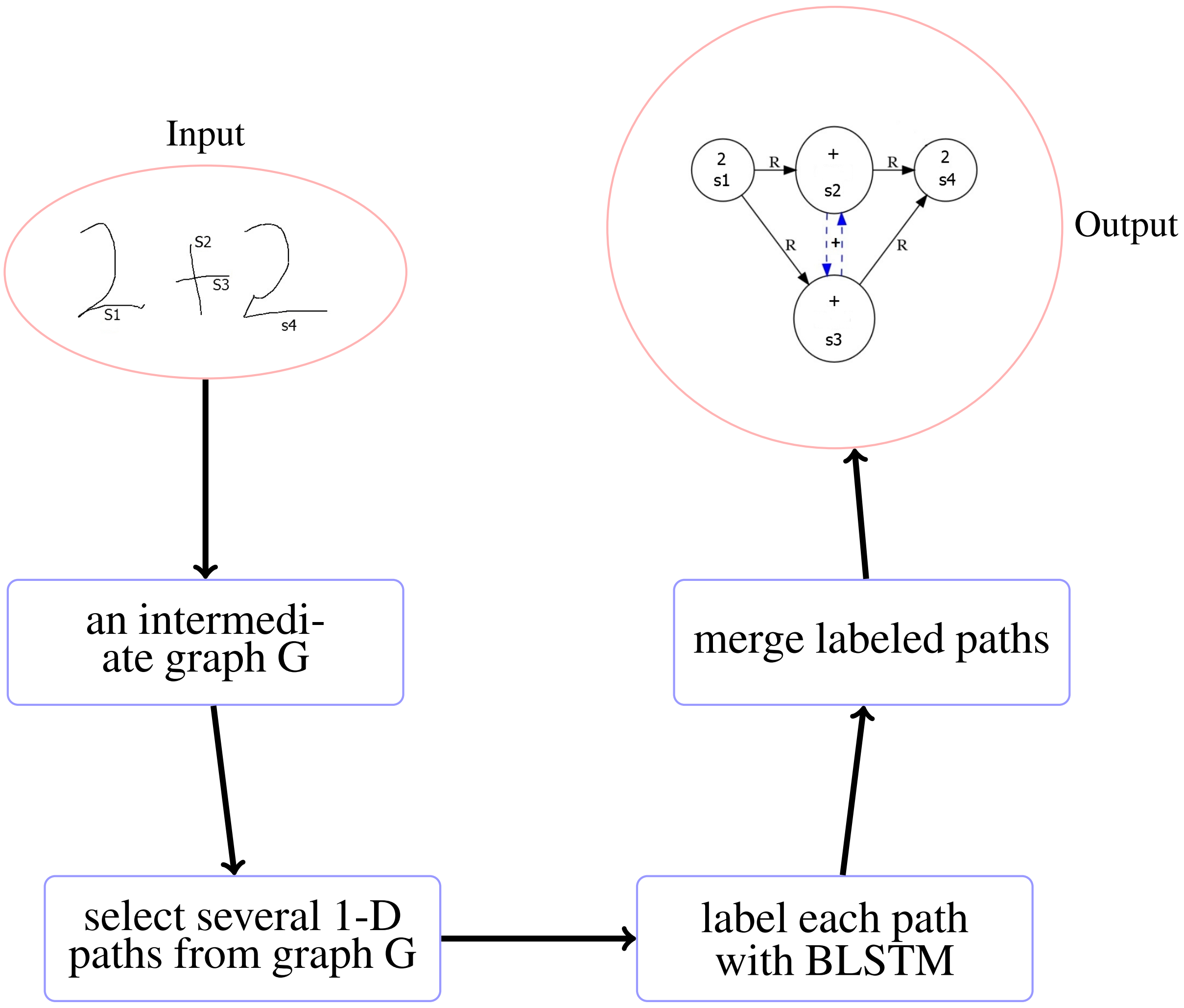


Figure 5.4 – Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME.

considering both the temporal and spatial information. In this graph, each node is a stroke and edges are added according to temporal or spatial properties. The derived graph is expected to have a high recall and a reasonable precision compared to the ground truth SLG. The remaining work is to label each node and each edge of the graph. To this end, several 1-D paths will be selected from the graph since the classifier model we are considering is a sequence labeler. The classical BLSTM-RNN model are able to deal with only sequential structure data. Next, we use the BLSTM classifier to label the selected 1-D paths. This stage consists of two steps, being the training and recognition process. Finally, we merge these labeled paths to build a complete stroke label graph with a strategy of setting different weights for them.

5.3 Detailed implementation

As explained in last section, the input data is available as a sequence of strokes $S = (s_0, ..., s_{n-1})$ (for $i < j$, we assume s_i has been entered before s_j) from which we would like to obtain the final SLG graph describing unambiguously the ME. In this part, we will introduce the recognition system step by step following the order within the framework.

5.3.1 Derivation of an intermediate graph G

In a first step, we will derive an intermediate graph G, where each node is a stroke and edges are added according to temporal or spatial properties. Based on the previous works on graph reprensentation that we reviewed in Section 5.1, we develop a new directed graph representation model. Some definitions regarding to the spatial relationships between strokes will be provided first.

Definition 5.1. The distance between two strokes s_i and s_j can be defined as the Euclidean distance between their closest points.

$$dist(s_i, s_j) = \min_{p \in s_i, q \in s_j} \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \tag{5.1}$$

It is the CPP distance mentioned in Section 5.1 as a matter of fact.

Definition 5.2. A stroke s_i is considered visible from stroke s_j if the bounding box of the straight line between their closest points does not cross the bounding box of any other stroke s_k.

For example in Figure 5.5, s_1 and s_3 can see each other because the bounding box of the straignt line between their closest points does not cross the bounding box of stroke s_2 and s_4. In [Muñoz, 2015] the visibility is defined by the straight line between their closest points does not cross any other stroke. We simplify it through replacing the stroke with its bounding box to reduce computation.

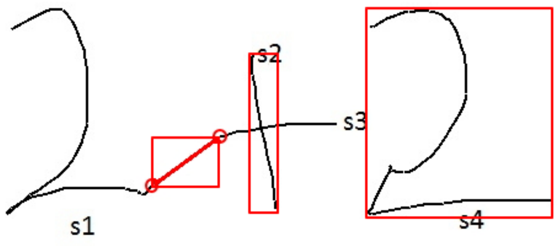


Figure 5.5 – Illustration of visibility between a pair of strokes. s_1 and s_3 are visible to each other.

Definition 5.3. For each stroke s_i, we define 5 regions ($R1, R2, R3, R4, R5$ shown in Figure 5.6). The center of the bounding box is taken as the reference point.

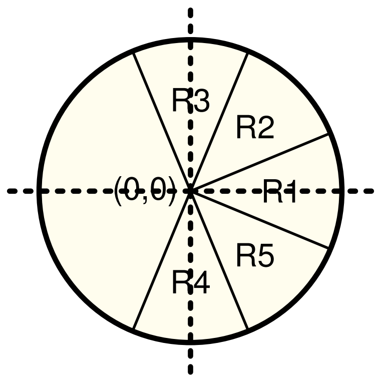


Figure 5.6 – Five directions for a stroke s_i. Point (0, 0) is the center of bounding box of s_i. The angle of each region is $\frac{\pi}{4}$.

As illustrated in Figure 5.6, point (0, 0) is the center of the bounding box of stroke s_i. The angle of each region is $\frac{\pi}{4}$. If the center of bounding box of s_j is in one of these five regions, for example $R1$ region, we can say s_j is in the $R1$ direction of s_i. The purpose of defining these 5 regions is to look for the $Above$, $Below$, Sup, Sub and $Right$ relationships between strokes in these 5 preferred directions, but not to recognize them.

Definition 5.4. 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 crossing strokes $S_{\text{cro}}(i) = \{s_{\text{cro1}}, s_{\text{cro2}}, ...\}$ from $\{s_{i+1}, ..., s_{n-1}\}$.
- the set of closest stroke $S_{\text{clo}}(i) = \{s_{\text{clo}}\}$ from $\{s_{i+1}, ..., s_{n-1}\} - S_{cro(i)}$.

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

- the set $S_{\text{vis}}(i)$ of the visible closest strokes in each of the five directions respectively from S - $\{s_i\} \bigcup S_{\text{cro}}(i) \bigcup S_{\text{clo}}(i)$. Here, the closeness of two strokes is decided by the distance between the centers of their bounding boxes, differently from Definition 5.1.

Edges from s_i to the $S_{\text{cro}}(i) \bigcup S_{\text{clo}}(i) \bigcup S_{\text{vis}}(i)$ will be added to G. Finally, we check if the edge from s_i to s_{i+1} (i from 0 to $n-2$) exists in G. If not, then add this edge to G to ensure that the path covering the sequence of strokes in the time order is included in G.

An example is presented in Figure 5.7. Mathematical expression $\frac{d}{dx}a^x$ is written with 8 strokes (Figure 5.7a). From the sequence of 8 strokes, the graph shown in Figure 5.7b is generated with the above mentioned method. Comparing the built graph with the ground truth (Figure 5.7c), we can see the difference in Figure 5.7d. All the ground truth edges are included in the generated graph except edges (blue ones in Figure 5.7d) from strokes 4 to 3 and from strokes 7 to 6. This flaw can be overcome as long as strokes 3 and 4 and the edge from strokes 3 to 4 are correctly recognized. Because if strokes 3 and 4 are recognized as belonging to the same symbol, the edge from strokes 4 to 3 can be completed automatically, as well as the edge from strokes 7 to 6. In addition, Figure 5.7d indicates the unnecessary edges (red edges) when matching the built graph to the ground truth. It is expected that the graph built could include the ground truth edges as many as possible, simultaneously contains the unnecessary edges as few as possible.

5.3.2 Graph evaluation

Hu evaluates the graph representation model by comparing the edges of the graph with ground truth edges at the stroke level[Hu, 2016]. The recall and precision rates are considered. In this section, we take a

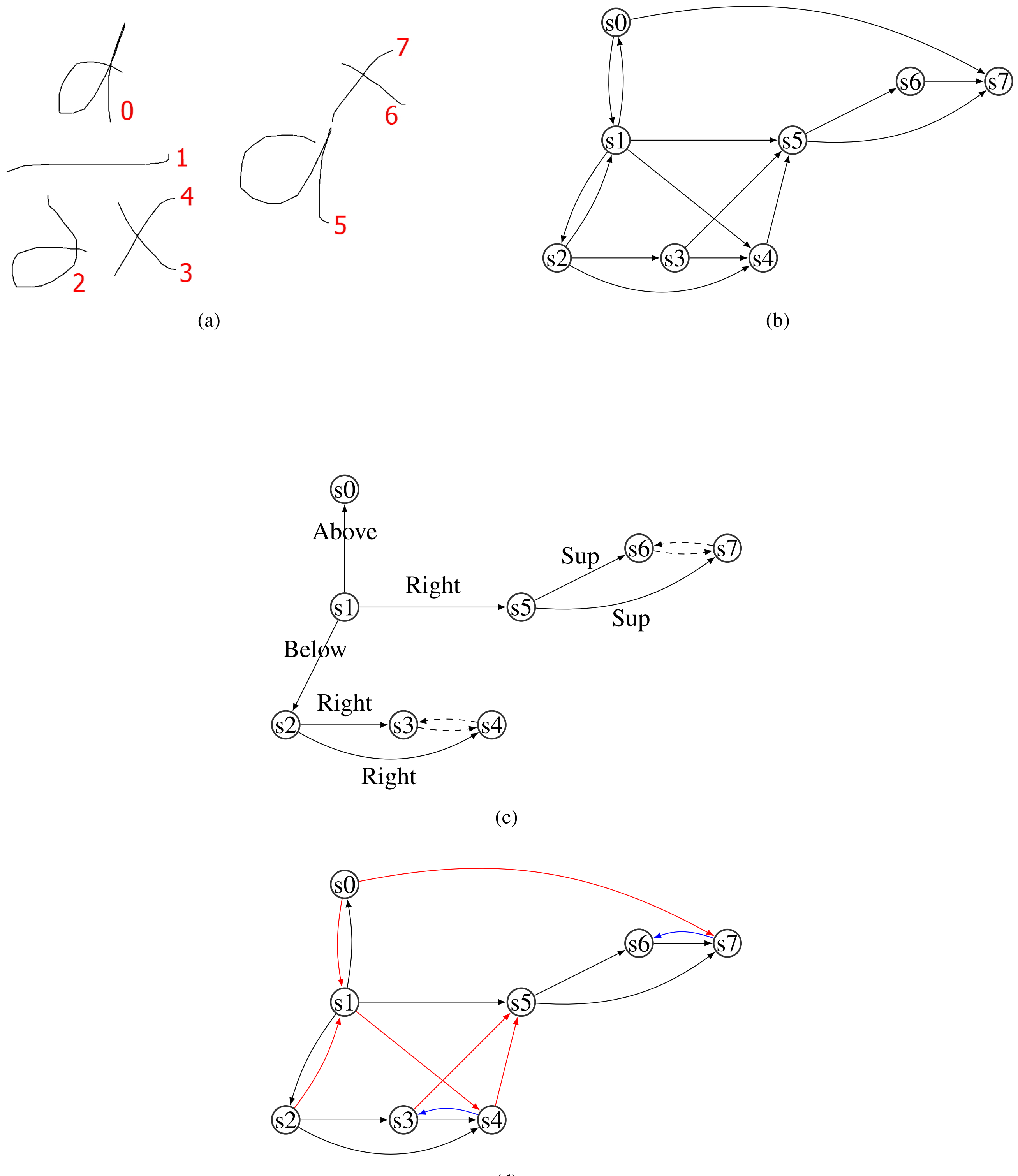


Figure 5.7 – (a)$\frac{d}{dx}a^x$ is written with 8 strokes; (b) the SLG built from raw input using the proposed method; (c) the SLG from ground truth; (d) illustration of the difference between the built graph and the ground truth graph, red edges denote the unnecessary edges and blue edges refer to the missed ones compared to the ground truth.

similar but more directive method, the same solution as introduced in Section 4.3.2. Specifically, provided the ground truth labels of the nodes and edges in the graph, we would like to see evaluation results at symbol and expression levels.

We reintroduce the evaluation criteria here as a kind reminder to readers: symbol segmentation ('Segments'), refers to a symbol that is correctly segmented whatever the label; 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 right relationship label.

Table 5.1 and Table 5.2 present the evaluation results of the graph construction on CROHME 2014 test set (provided the ground truth labels) at the symbol and expression level respectively. We re-show the evaluate results, already given in Tables 4.1 and 4.2, of time graph as a reference to the new graph. Due to the delayed strokes, time graph miss a small part of segmentation edges. Thus around 0.27% symbols are wrongly segmented. The new graph achieves 100% recall rate and 99.99% precision rate on segmentation task (0.01% error is resulted from a small error in data set, not the model itself). These figures evidence that the new model could handle the case of delayed strokes. With regards to relationship recognition task, time graph model misses about 25% relationships. The new graph catches 93.48% relationships. Compared to time graph, there is a great improvement in relationship representation. However, owing to the missed 6.52% relationships, only 67.65% expressions are correctly recognized as presented in Table 5.2. These values will be upper bounds for the recognition system based on this graph model.

Table 5.1 – The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels of the nodes and edges of the built graph).

Model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
time graph	99.73	99.45	99.72	99.44	75.54	99.27
new graph	100.00	99.99	99.99	99.98	93.48	99.95

Table 5.2 – The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels of the nodes and edges of the built graph).

Model	correct (%)	<= 1 error	<= 2 errors	<= 3 errors
time graph	34.11	40.94	50.51	58.25
new graph	67.65	76.70	85.76	90.74

5.3.3 Select paths from G

The final aim of our work is to build the SLG of 2-D expression. The proposed solution is carried out with merging several 1-D paths from G. These paths are expected to cover all the nodes and as many as the edges of the ground truth SLG (at least the edges of the ground truth SRT). With the correctly recognized node and edge labels, we have the possibility to build a correct 2-D expression. Obviously, a single 1D path is not able to cover all these nodes and edges, except in some simple expressions. We have explained this point in detail in Chapter 4. This section will explain how we generate several paths from the graph, enough different to cover the SRT, and then how to merge the different decisions in a final graph.

A path in G can be defined as $\Phi_i = (n_0, n_1, n_2, ..., n_e)$, where n_0 is the starting node and n_e is the end node. The node set of Φ_i is $n(\Phi_i) = \{n_0, n_1, n_2, ..., n_e\}$ and the edge set of Φ_i is $e(\Phi_i) = \{n_0 \rightarrow n_1, n_1 \rightarrow n_2, ..., n_{e-1} \rightarrow n_e\}$.

Two types of paths are selected in this chapter: $time$ path and $random$ path. $time$ path starts from the first input stroke and ends with the last input stroke following the time order. For example, in Figure 5.7d, the $time$ path is $(0, 1, 2, 3, 4, 5, 6, 7)$. Then, we consider several additional random paths. To ensure a good coverage of the graph we guide the random selection choosing the less visited nodes and edges (give higher priority to less visited ones). The algorithm of selecting $random$ path is as follows:

(1) Initialize $T_n = 0$, $T_e = 0$. T_n records the number of times that node n has been chosen as starting node, T_e records the number of times of that edge e has been used in chosen paths;

(2) Update $T_n = T_n + 1$, $T_e = T_e + 1$ of all the nodes and edges in $time$ path;

(3) Randomly choose one node N from the nodes having the minimum T_n, update $T_N = T_N + 1$;

(4) Find all the edges connected to N, randomly choose one from the edges having the minimum T_e, denoted as E, update $T_E = T_E + 1$; if no edge found, finish.

(5) Reset N as the to node of E, go back to step 4.

One $random$ path could be like $(1, 5, 6, 7)$.

5.3.4 Training process

Each path, $time$ or $random$, is handled independently during the training process as a training sample. In Chapter 4, we introduced the technique related to training a time path, such as how to feed BLSTM inputs (Section 4.2.1), extracting features(Section 4.2.2) and local CTC training method (Section 4.2.3); the same training process is kept for random paths in this chapter.

Totally, we have 3 types of BLSTM classifiers trained with respectively only $time$ path, only $random$ paths and $time + random$ paths. More related contents could be found in the experimental section of this chapter.

5.3.5 Recognition

Since we use local CTC technique in the training process in this work, naturally the recognition stage should be performed on stroke ($strokeD$ and $strokeU$) level. As explained previously, to build the SLG, we also need to assign one single label to each stroke. Considering these two causes, a pooling strategy is required to go from the point level to the stroke level since for each point or time step, the network outputs the probabilities of this point belonging to different classes. We proposed two kinds of decoding methods based on stroke level (**maximum decoding**, **local CTC decoding**) and tested the effects of them in Chapter 4. According to the evaluation results, **maximum decoding** is a better choice for its low computation and same level effectiveness as **local CTC decoding**.

With the Equation 4.16, we can compute P_c^s, the cumulative probability of outputting the c^{th} label for stroke s. Then we sort the normalized P_c^s and only keep the top n probable labels (excluding $blank$) with the accumulative probability ≥ 0.8. Note that n is maximum 3 even though the accumulative probability of top 3 labels is not up to 0.8.

5.3.6 Merge paths

Each stroke belongs to at least one path, but possibly to several paths. Hence, several recognition results can be available for a single stroke. At this stage, we propose to compute the probability $P_s(l)$ to assign the label l to the stroke s by summing all path Φ_i with the formula:

$$P_s(l) = \frac{\sum_{\Phi_i} W_{\Phi_i} * P_l^{(\Phi_i,s)} \mathbb{1}_A(s) \mathbb{1}_{label(\Phi_i,s)}(l)}{\sum_{\Phi_i} W_{\Phi_i} \mathbb{1}_A(s)} \tag{5.2}$$

$$A = n(\Phi_i) \cup e(\Phi_i) \tag{5.3}$$

$$\mathbb{1}_M(m) = \begin{cases} 1 & \text{if } m \in M \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

W_{Φ_i} is the weight set for path Φ_i and $label(\Phi_i, s)$ is the set of candidate labels for stroke s from path Φ_i, $1 \leq |label(s, \Phi_i)| \leq 3$. If stroke s exists in path Φ_i, but $l \notin label(s, \Phi_i)$, in this case, $P_l^{(\Phi_i,s)}$ is 0. The classifier answers that there is no possibility to output label l for stroke s from path Φ_i. We still add W_{Φ_i} into the normalized factor of $P_s(l)$. If stroke s does not exist in path Φ_i, the classifier's answer to stroke s is unknown. And we should not take into account this path Φ_i. Thus, W_{Φ_i} will not be added into the normalized factor of $P_s(l)$. After normalization, the label with the maximum probability is selected for each stroke.

As shown in Figure 5.8, we consider merging 3 paths Φ_1, Φ_2, Φ_3. Stroke s only belongs to path Φ_1, Φ_2. In path Φ_1, the candidate labels for stroke s are a, b, c while in path Φ_2, the candidate labels are b, c, d. The

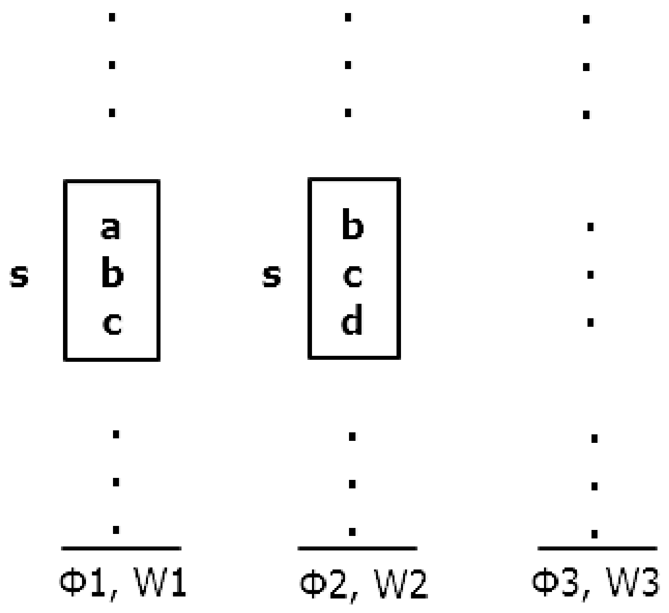


Figure 5.8 – Illustration of the strategy for merge.

probability of assigning a to stroke s is computed as:

$$P_s(a) = \frac{W_{\Phi_1} * P_a^{\Phi_1,s} + W_{\Phi_2} * 0 + W_{\Phi_3} * 0}{W_{\Phi_1} + W_{\Phi_2} + W_{\Phi_3} * 0} \tag{5.5}$$

Stroke s is not covered by path Φ_3, thus W_{Φ_3} will not be added into the normalized factor. The probability of outputting label a for stroke s in path Φ_2 is 0. Furthermore, the probability of assigning b to stroke s is computed as:

$$P_s(b) = \frac{W_{\Phi_1} * P_b^{\Phi_1,s} + W_{\Phi_2} * P_b^{\Phi_2,s} + W_{\Phi_3} * 0}{W_{\Phi_1} + W_{\Phi_2} + W_{\Phi_3} * 0} \tag{5.6}$$

In conclusion, we combine the recognition results from different paths (while in Chapter 4, only one path is used), and then select for each node or edge the most probable label. Afterwards, an additional process will be carried out in order to build a valid LG, i.e. adding edges, in the same way as what have done in Chapter 4. We first look for the segments (symbols) using connected component analysis: a connected component where nodes and edges have the same label is a symbol. With regards to the relationship between two symbols, we choose the label having the maximum accumulative probability among the edges between two symbols. Then, 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.

5.4 Experiments

In the CROHME 2014 data set, there are 8834 expressions for training and 982 expressions for test. Likewise, we divide 8834 expressions for training (90%) and validation (10%); use CROHME 2014 test set for test. Based on the RNNLIB library [1], the recognition system is developed by merging multiple paths. For each training process, the network having the best CTC error on validation data set is saved. Then, we evaluate this network on the test data set. The Label Graph Evaluation library (LgEval) [Mouchère et al., 2014] is used to analyze the recognition output. The specific configuration for network architecture is the same as the one we set in Section 4.3. This configuration has obtained good results in both handwritten text recognition [Graves et al., 2009] and handwritten math symbol classification [Álvaro et al., 2013, 2014a].

The size of the input layer is 5 (5 local features, as same as Chapter 4) while the output layer size in this experiment is 109 (101 symbol classes + 6 relationships + $NoRelation$ + $blank$). For each expression, we extract the $time$ path and 6 (or 10) $random$ paths. Totally, 3 types of classifiers are trained: the first one with only $time$ path (denoted as CLS_T, actually it is the same classifier as in Chapter 4); the second BLSTM network is trained with only 6 $random$ paths (denoted as CLS_{R6}, for 10 $random$ paths we use CLS_{R10}); and the third classifier uses $time$ + 6 $random$ paths (denoted as CLS_{T+R6}). We train these 3 types of classifiers to see the effect of different training content on recognition result, also the impact of the number of paths. We use these 4 different classifiers (CLS_T, CLS_{R6}, CLS_{R10}, CLS_{T+R6}) to label the different types of paths extracted from the test set as presented in Table 5.3 (exp.1, uses CLS_T to label $time$ path; exp.2, uses CLS_T to label both $time$ path and $random$ paths; exp.3, uses CLS_T to label $time$ path and CLS_R to label $random$ paths; exp.4, uses CLS_{T+R6} to label both $time$ path and $random$ paths; exp.5, use CLS_T to label $time$ path and CLS_{R10} to label $random$ paths;). In exp.1, only the labeled $time$ path is used to build a 2-D expression, actually it is the same case carried out in Chapter 4. For exp.(2 3 4), $time$ path and 6 $random$ paths are merged to construct a final graph. $time$ path and 10 $random$ paths are contained in exp.5. The weight of $time$ path is set to 0.4 and each random path is 0.1 [2].

Table 5.3 – Illustration of the used classifiers in the different experiments depending of the type of path.

exp.	Path Type	
	Time	Random
1	CLS_T	
2	CLS_T	CLS_T
3	CLS_T	CLS_{R6}
4	CLS_{T+R6}	CLS_{T+R6}
5	CLS_T	CLS_{R10}

The evaluation results at symbol level are provided in Table 5.4 including recall ('Rec.') and precision ('Prec.') rates for symbol segmentation ('Segments'), symbol segmentation and recognition ('Seg+Class'),

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

2. The weights are manually optimized. We tested several different weight assignments, and then choose the best one among them.

spatial relationship classification ('Tree Rels.'). A correct spatial relationship between two symbols requires that both symbols are correctly segmented and with the right relationship label. As presented, the results for 'Segments' and 'Seg+Class' do not show a big difference among exp.(1 2 3 4). It can be explained by the fact that $time$ path is enough to give good results and $random$ paths contributes little. With regard to 'Tree Rels.', 'Rec.' of exp.(2 3 4) is improved compared to exp.1 because $random$ paths catch some ground truth edges which are missed in $time$ path; but 'Prec.' rate declines which means that $random$ paths also cover some edges which are not in ground truth LG. Unfortunately, these extra edges are not labeled as $NoRelation$. Among (1 2 3 4) experiments, exp.3 outperforms others for all the items. Thus, it is a better strategy to use CLS_T for labeling $time$ path and use CLS_R for $random$ path. Our results are comparable to the results of CROHME 2014 because the same training and testing data sets are used. The second part of Table 5.4 gives the symbol level evaluation results of the participants in CROHME 2014 sorting by the recall rate for correct symbol segmentation. The best 'Rec.' of 'Segments' and 'Seg+Class' reported by CROHME 2014 are 98.42% and 93.91% respectively. Ours are 92.77% and 85.17%, both ranked 3 out of 8 systems (7 participants in CROHME 2014). Our solution presents competitive results on symbol recognition task and segmentation task.

Table 5.5 shows the recognition rates at the global expression level with no error, and with at most one to three errors in the labels of LG. The recognition rate with no error on CROHME 2014 test set is 13.02% in exp.3. The best one and worst one reported by CROHME 2014 are 62.68% and 15.01%. With regard to the recognition rate with ≤ 3 errors, 4 participants are between 27% and 37% and our result is 35.71%.

Table 5.4 – The symbol level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results.

exp.	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
1	92.30	85.15	83.70	77.22	58.40	**74.27**
2	92.32	85.28	84.37	77.94	**67.79**	58.34
3	**92.77**	**85.99**	**85.17**	**78.95**	**67.79**	**67.33**
4	91.51	83.48	82.96	75.67	66.95	62.06
5	**92.97**	**86.35**	**85.73**	**79.63**	**72.00**	**64.01**
system	CROHME 2014 participant results					
III	98.42	98.13	93.91	93.63	94.26	94.01
I	93.31	90.72	86.59	84.18	84.23	81.96
VII	89.43	86.13	76.53	73.71	71.77	71.65
V	88.23	84.20	78.45	74.87	61.38	72.70
IV	85.52	86.09	76.64	77.15	70.78	71.51
VI	83.05	85.36	69.72	71.66	66.83	74.81
II	76.63	80.28	66.97	70.16	60.31	63.74

Among (1 2 3 4) experiments, exp.3 outperforms others for all the items. Compared to exp.1 where only $time$ path is considered, we see an increase on recall rate of 'Tree Rels.' but meanwhile a decrease on precision rate of 'Tree Rels.' in exp.3. As only 6 $random$ paths is used in it, we would like to see if more $random$ paths could bring any changes. We carry out another experiment, exp.5, where 10 $random$ paths is used to train classifier CLS_R. The evaluation results on symbol level and expression level are provided respectively in Table 5.4 and Table 5.5. As shown, when we consider more $random$ paths, the recall rate of 'Tree Rels.' keeps on increasing but precision rate of 'Tree Rels.' is decreasing. Thus, at the expression level, the recognition rate remains the same level as the experiment with 6 $random$ paths.

To illustrate these results, we reconsider the 2 test samples ($a \geq b$ and $44 - \frac{4}{4}$) recognized with the system of exp.3. In last chapter where we just use the single $time$ path, $a \geq b$ was correctly recognized and for $44 - \frac{4}{4}$, the $Right$ relationship from the minus symbol to fraction bar was omitted in the modeling

Table 5.5 – The expression level evaluation results on CROHME 2014 test set, including the experiment results in this work and CROHME 2014 participant results

exp.	correct (%)	$\leq$ 1 error	$\leq$ 2 errors	$\leq$ 3 errors
1	11.80	19.33	26.55	31.43
2	8.55	16.89	23.40	29.91
3	**13.02**	**22.48**	**30.21**	**35.71**
4	11.19	19.13	26.04	31.13
5	**13.02**	**21.77**	**30.82**	**36.52**
system	CROHME 2014 participant results			
III	62.68	72.31	75.15	76.88
I	37.22	44.22	47.26	50.20
VII	26.06	33.87	38.54	39.96
VI	25.66	33.16	35.90	37.32
IV	18.97	28.19	32.35	33.37
V	18.97	26.37	30.83	32.96
II	15.01	22.31	26.57	27.69

stage, likewise the $Above$ relationship from the fraction bar to the numerator 4. In addition, the relation from the minus symbol to the numerator 4 was wrongly recognized as $Right$ and it should be labeled as $NoRelation$.

In this chapter, $a \geq b$ is recognized correctly also (Figure 5.9). We present the the derived graph in Figure 5.9b. Then from the graph, we extract the $time$ path and 6 $random$ paths. In this example, all the nodes and edges of Figure 5.9b are included in the extracted 7 paths. After merging the results of 7 paths with the Equation 5.2, we can get the labeled graph illustrated as Figure 5.9c. The edge from stroke 0 to 1 is wrongly labeled as Sup. Next, we carry out the post process stage. The segments (symbols) are decided using connected component analysis: 3 symbols $(a, \geq, b)$ in this expression. With regards to the relationship between a and $\geq$, we have 2 candidates Sup with the probability 0.604 and $Right$ with the probability 0.986. With the strategy of choosing the label having the maximum accumulative probability, the relationship between a and $\geq$ is $Right$ then. After post process, we construct a correct SLG provided in Figure 5.9d.

The recognition result for $44 - \frac{4}{4}$ is presented in Figure 5.10. From the handwritten expression (Figure 5.10a), we could derive a graph presented in (Figure 5.10b). Then, we extract paths from the graph, and label them, finally merge the labeled paths to built a labeled graph. Figure 5.10c provides the built SLG from which we can see several extraneous edges appear owing to multiple paths and in this sample they are all recognized correctly as $NoRelation$. We remove these $NoRelation$ edges to have a intuitive judgment on the recognition result (Figure 5.10d).As can be seen, the $Right$ relationship from the minus symbol to fraction bar is missed. This error comes from the graph representation stage where we find no edge from stroke 2 to 4. Both stroke 3 and 4 are located in $R1$ region of stroke 2, but stroke 3 is closer to stroke 2 than stroke 4. Thus, we miss the edge from stroke 2 to 4 at the graph representation stage, and naturally miss the $Right$ relationship from the minus symbol to fraction bar in the built SLG. This error can be overcome by searching for a better graph model or some post process strategies regarding to the connected SLG.

As discussed above, our solution presents competitive results on symbol recognition task and segmentation task, but not on relationship detection and recognition task. Compared to the work of Chapter 4, the solution in this chapter presents improvements on recall rate of 'Tree Rels.' but at the same time decreases on precision rate of 'Tree Rels.' Thus, at the expression level, the recognition rate remains the same level as the solution with single path. One of the intrinsic causes is that even though several paths from one expression is considered in this system, the BLSTM model processes each path separately which means the model only could access the contextual information in one path during training and recognition stages. Obviously it conflicts with the real case that human beings recognize the raw input using the entire contex-

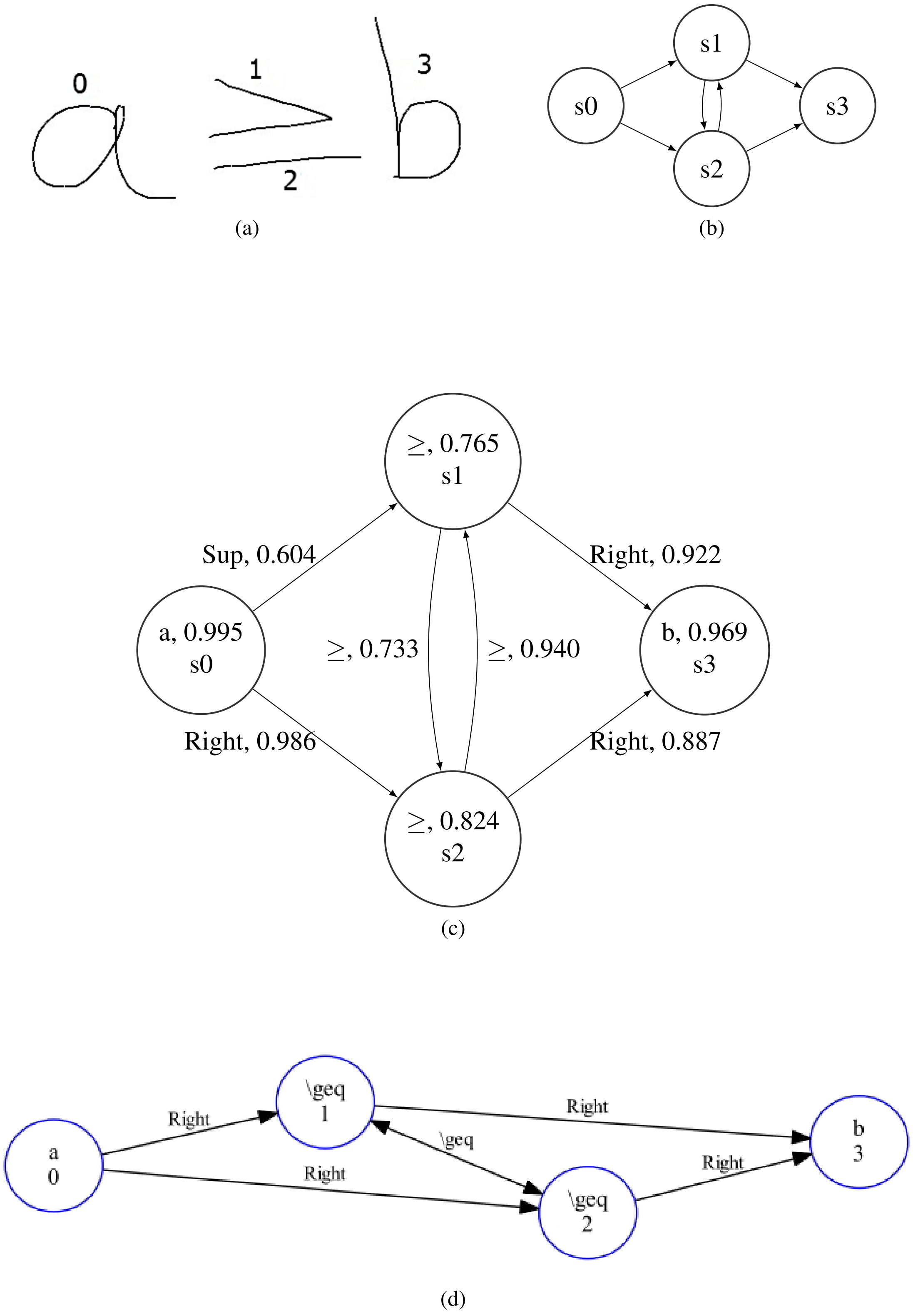
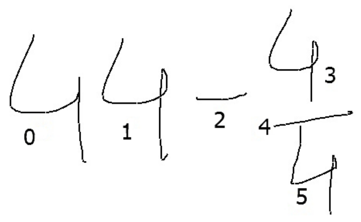
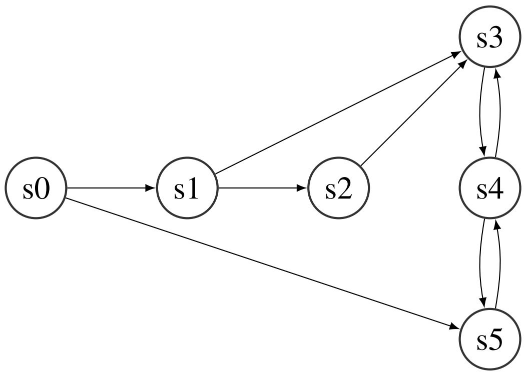


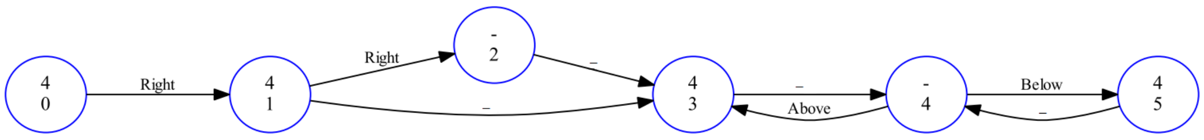
Figure 5.9 – (a) $a \geq b$ written with four strokes; (b) the derived graph from the raw input; (c) the labeled graph (provided the label and the related probability) with merging 7 paths; (d) the built SLG after post process, all labels are correct.



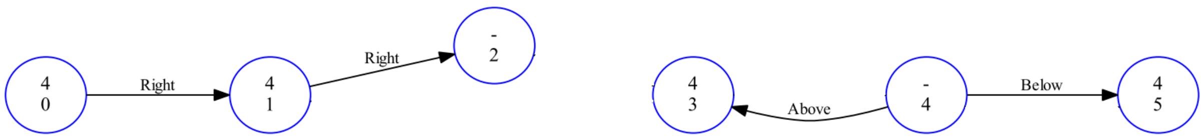
(a)



(b)



(c)



(d)

Figure 5.10 – (a) $44 - \frac{4}{4}$ written with six strokes; (b) the derived graph; (c) the built SLG by merging several paths; (d) the built SLG with $NoRelation$ edges removed.

tual information. In the coming chapter, we will search for a model which could take into account more contextual knowledge at one time instead of just the content limited in one single path.

5.5 Discussion

We recognize 2-D handwritten mathematical expressions by merging multiple 1-D labeled paths in this chapter. Given an expression, we propose an algorithm to generate an intermediate graph using both temporal and spatial information between strokes. Next from the derived graph, different types of paths are selected and later labeled with the strong sequence labeler—BLSTM. Finally, we merge these labeled paths to build a 2-D math expression. The proposal presents competitive results on symbol recognition task and segmentation task, promising results on relationship recognition task. Compared to the work of Chapter 4, the solution in this chapter presents improvements on recall rate of ‘Tree Rels.’ but at the same time decreases on precision rate of ‘Tree Rels.’ Thus, at the expression level, the recognition rate remains the same level as the solution with single path.

Currently, even though several paths from one expression is considered in this system, in essential the BLSTM model deals with each path isolatedly. The classical BLSTM model could access information from past and future in a long range but the information outside the single sequence is of course not accessible to it. In fact, it conflicts with the real case that human beings recognize the raw input using the entire contextual information. As shown in our experiments, it is laborious to solve a 2-D problem with a chain-structured model. Thus, we would like to develop a tree-structured neural network model which could handle directly the structure not limited to a chain. With the new neural network model, we could take into account more contextual information in a tree instead of a single 1D path.

6

Mathematical expression recognition by merging multiple trees

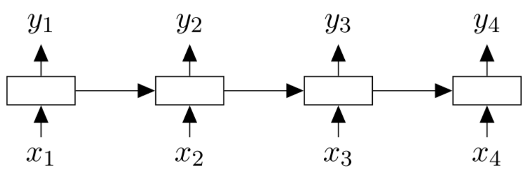
In Chapter 5, we concluded that it is hard to use the classical chain-structured BLSTM to solve the problem of recognizing mathematical expression which is a tree structure. In this chapter, we extend the chain-structured BLSTM to tree structure topology and apply this new network model for online math expression recognition.

Firstly, we provide a short overview with regards to the Non-chain-structured LSTM. Then, we propose in Section 6.2 a new neural network model named tree-based BLSTM which seems to be appropriate for this recognition problem. Section 6.3 globally introduces the framework of mathematical expression recognition system based on tree-based BLSTM. Hereafter, we focus on the specific techniques involved in this system in Section 6.4. Finally, experiments and discussion parts are covered in Section 6.5 and Section 6.7 respectively.

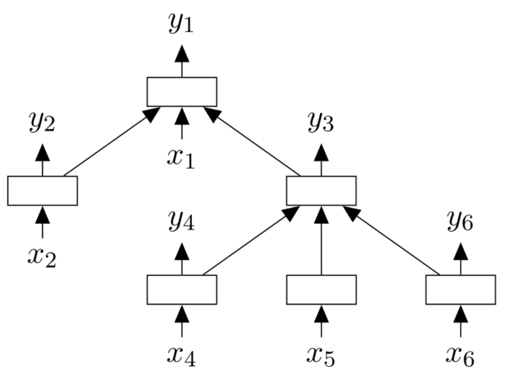
6.1 Overview: Non-chain-structured LSTM

A limitation of the classical LSTM network topology is that they only allow for sequential information propagation (as shown in Figure 6.1a) 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 using n recurrent connections (one for each of the cell's previous states along every dimension) 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 [Graves et al., 2012]. MDLSTM model exhibits great performances on offline handwriting recognition tasks where the input is an image [Graves and Schmidhuber, 2009, Messina and Louradour, 2015, Bluche et al., 2016, Maalej and Kherallah, 2016, Maalej et al., 2016].

In [Tai et al., 2015], 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 (Figure 6.1b). Since 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 [Tai et al., 2015], [Zhu et al., 2015] explored the similar idea



(a)



(b)

Figure 6.1 – (a) A chain-structured LSTM network; (b) A tree-structured LSTM network with arbitrary branching factor. Extracted from [Tai et al., 2015].

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 [Zhu et al., 2016], possessing the ability to incorporate external semantics including non-compositional or holistically learned semantics.

6.2 The proposed Tree-based BLSTM

This section will be focused on Tree-based BLSTM. Different with the tree structures depicted in [Tai et al., 2015, Zhu et al., 2015], we devote it to the kind of structures presented in Figure 6.2 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 process 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.

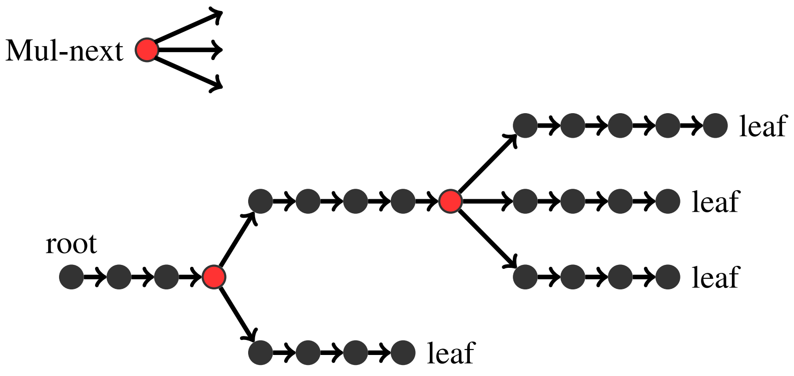


Figure 6.2 – A tree based structure for chains (from root to leaves).

From root to leaves. There are 2 special nodes (red) having more than one next node in Figure 6.2. 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 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*.

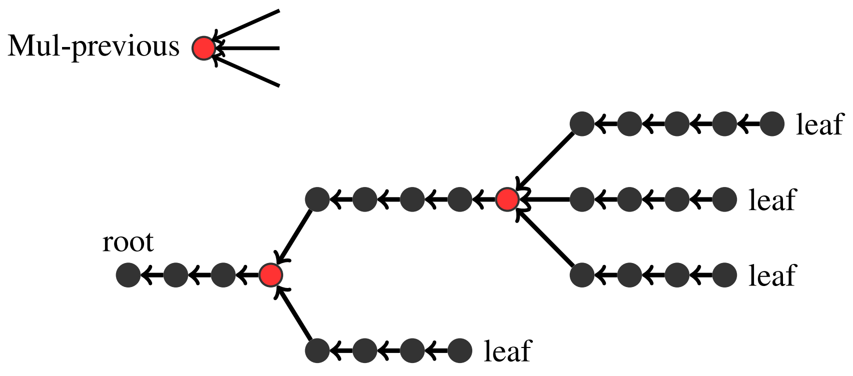


Figure 6.3 – A tree based structure for chains (from leaves to root).

From leaves to root. Suppose all the arrows in Figure 6.2 are reversed, we have the new structure which is actually beyond a tree in Figure 6.3. The 2 red nodes are still special cases because they have more than one previous node. 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 Chapter 3 and [Graves et al., 2012] 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 referring to the input, forget and output gate. The subscript c refers to one of the C cells. Thus, the peep-hole 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 $\mathrm{Pr}(n)$ 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 which have been recalled in Chapter 3.

The forward propagation of *Mul-previous node*

Input gates

$$a_\varsigma^n = \sum_{i=1}^{I} w_{i\varsigma} x_i^n + \sum_{h=1}^{H} w_{h\varsigma} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\varsigma} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} s_c^p} \tag{6.1}$$

$$b_\varsigma^n = f(a_\varsigma^n) \tag{6.2}$$

Forget gates

$$a_\phi^n = \sum_{i=1}^{I} w_{i\phi} x_i^n + \sum_{h=1}^{H} w_{h\phi} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\phi} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} s_c^p} \tag{6.3}$$

$$b_\phi^n = f(a_\phi^n) \tag{6.4}$$

Cells

$$a_c^n = \sum_{i=1}^{I} w_{ic} x_i^n + \sum_{h=1}^{H} w_{hc} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} b_h^p} \tag{6.5}$$

$$s_c^n = b_\phi^n \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} s_c^p} + b_\varsigma^n g(a_c^n) \tag{6.6}$$

Output gates

$$a_\omega^n = \sum_{i=1}^{I} w_{i\omega} x_i^n + \sum_{h=1}^{H} w_{h\omega} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} b_h^p} + \sum_{c=1}^{C} w_{c\omega} s_c^n \tag{6.7}$$

$$b_\omega^n = f(a_\omega^n) \tag{6.8}$$

Cell Outputs

$$b_c^n = b_\omega^n h(s_c^n) \tag{6.9}$$

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

We define

$$\epsilon_c^n = \frac{\partial L}{\partial b_c^n} \qquad \epsilon_s^n = \frac{\partial L}{\partial s_c^n} \qquad \delta_i^n = \frac{\partial L}{\partial a_i^n} \tag{6.10}$$

Then

$$\epsilon_c^n = \sum_{k=1}^{K} w_{ck} \delta_k^n + \sum_{g=1}^{G} w_{cg} \boxed{\sum_{e}^{|\mathrm{Ne}(n)|} \delta_g^e} \tag{6.11}$$

Output gates

$$\delta_w^n = f'(a_w^n) \sum_{c=1}^{C} h(s_c^n) \epsilon_c^n \tag{6.12}$$

States

$$\begin{aligned} \epsilon_s^n = b_w^n h'(s_c^n) \epsilon_c^n + \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} b_\phi^e \sum_{e=1}^{|\mathrm{Ne}(n)|} \epsilon_s^e} \\ + w_{c\varsigma} \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} \delta_\varsigma^e} + w_{c\phi} \boxed{\sum_{e=1}^{|\mathrm{Ne}(n)|} \delta_\phi^e} + w_{c\omega} \delta_\omega^n \end{aligned} \tag{6.13}$$

Cells

$$\delta_c^n = b_\varsigma^n g'(a_c^n) \epsilon_s^n \tag{6.14}$$

Forget gates

$$\delta_\phi^n = f'(a_\phi^n) \sum_{c=1}^{C} \boxed{\sum_{p=1}^{|\mathrm{Pr}(n)|} s_c^p \epsilon_s^n} \tag{6.15}$$

Input gates

$$\delta_\varsigma^n = f'(a_\varsigma^n) \sum_{c=1}^{C} g(a_c^n) \epsilon_s^n \tag{6.16}$$

6.3 The framework

We would apply the proposed tree-based BLSTM model for online mathematical expression recognition. This section provides a general view of the recognition system (Figure 6.4). Similar to the framework proposed in Chapter 5, we first drive an intermediate graph from the raw input. Then, instead of 1-D paths, we consider from the graph deriving trees which will be labeled by tree-based BLSTM model as a next step. In the end, these labeled trees will be merged to build a stroke label graph.

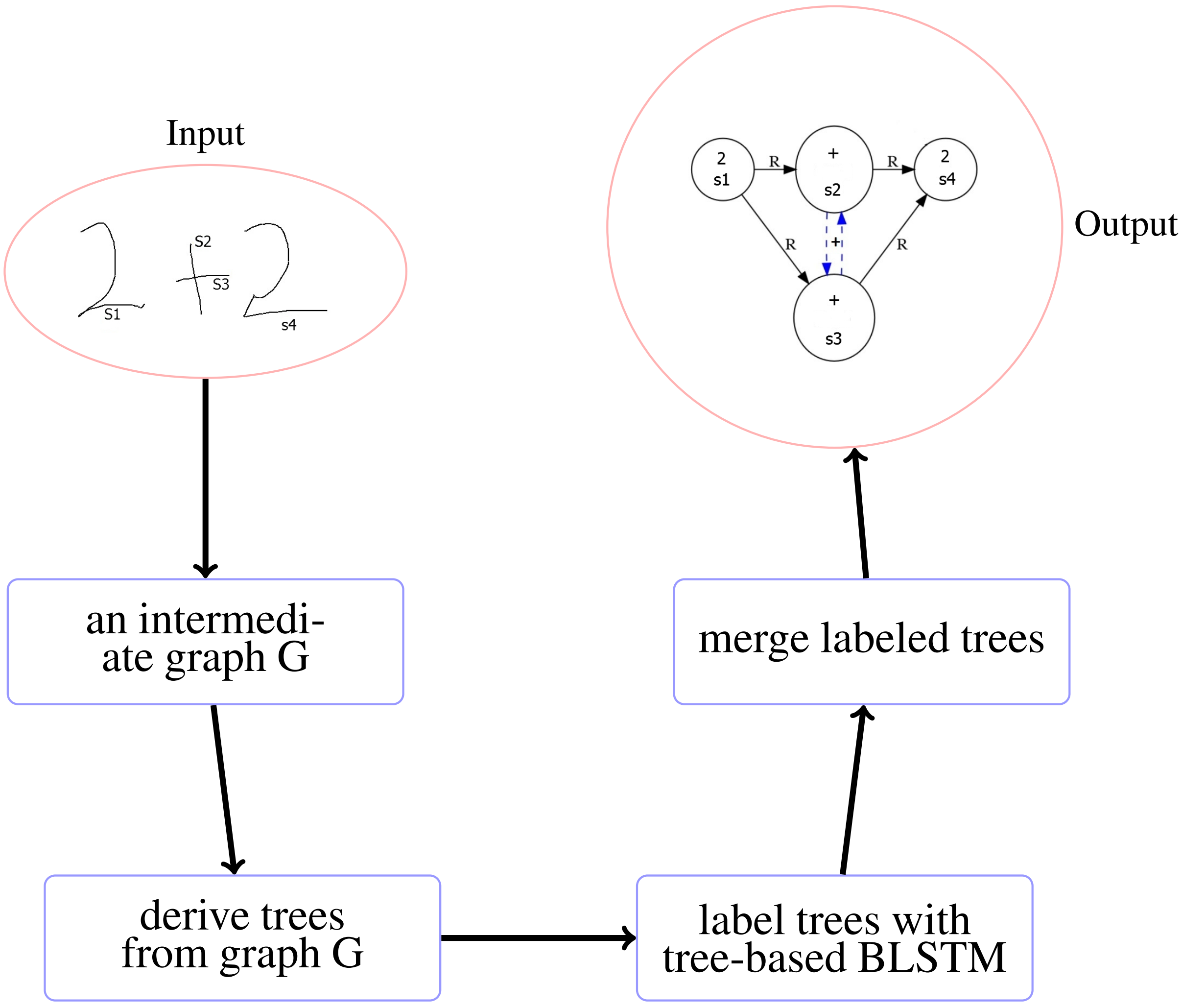


Figure 6.4 – Illustration of the proposal that uses BLSTM to interpret 2-D handwritten ME.

6.4 Tree-based BLSTM for online mathematical expression recognition

In this section, each step illustrated in Figure 6.4 will be elaborated in the text. The input data is available as a sequence of strokes S from which we would like to obtain the final LG graph describing unambiguously the ME. Let $S = (s_0, ..., s_{n-1})$, where we assume s_i has been written before s_j for $i < j$.

6.4.1 Derivation of an intermediate graph G

In a first step, we will derive an intermediate graph G where each node is a stroke and edges are added according to temporal or spatial relationships between strokes. In fact, we already introduced a graph representation model and evaluated it in Chapter 5. The evaluation results showed that around 6.5% relationships are missed compared to the ground truth graph. In this Section, we are aiming to improve the graph model to reduce the quantity of missed relationships. Similarly, we provide several definitions related to the graph building first.

Definition 6.1. A stroke s_i is considered visible from stroke s_j if the straight line between between their closest points does not cross any other stroke s_k.

For example, s_1 and s_3 can see each other because the straignt line between their closest points does not cross stroke s_2 or s_4 as shown in Figure 6.5. This definition is the same as the one used in [Muñoz, 2015]. Compared to Definition 5.2 where we replaced the stroke with its bounding box to reduce computation, the current one is more accurate.

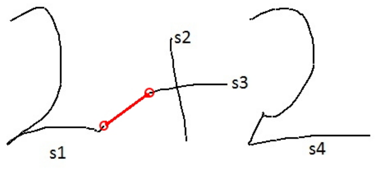


Figure 6.5 – Illustration of visibility between a pair of strokes. s_1 and s_3 are visible to each other.

Definition 6.2. For each stroke s_i, we define 5 regions ($R1, R2, R3, R4, R5$ shown in Figure 6.6) of it. The center of the bounding box of stroke s_i is taken as the reference point $(0, 0)$.

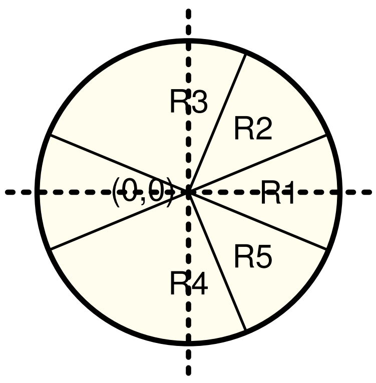


Figure 6.6 – Five regions for a stroke s_i. Point $(0, 0)$ is the center of bounding box of s_i. The angle range of $R1$ region is $[-\frac{\pi}{8}, \frac{\pi}{8}]$; $R2 : [\frac{\pi}{8}, \frac{3*\pi}{8}]$; $R3 : [\frac{3*\pi}{8}, \frac{7*\pi}{8}]$; $R4 : [-\frac{7*\pi}{8}, -\frac{3*\pi}{8}]$; $R5 : [-\frac{3*\pi}{8}, -\frac{\pi}{8}]$.

The purpose of defining these 5 regions is to look for the $Right$, $Supscript$, $Above$, $Below$ and $Subscript$ relationships between strokes. If the center of bounding box of s_j is located in one of five regions of stroke s_i, for example $R1$ region, we say s_j is in the $R1$ direction of s_i. In Definition 5.3, the angle of each region is $\frac{\pi}{4}$. Here, 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 6.3. 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):

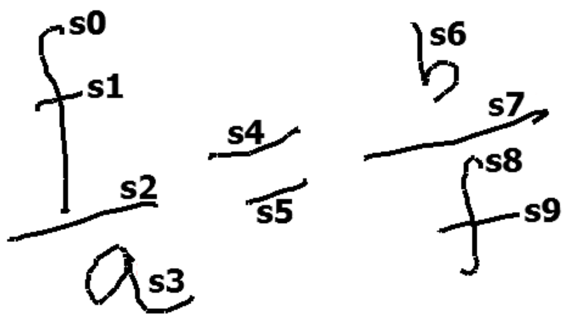
- the set of crossing future strokes $S_{\text{cro}}(i) = \{s_{\text{cro1}}, s_{\text{cro2}}, ...\}$ from $\{s_{i+1}, ..., s_{n-1}\}$.

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

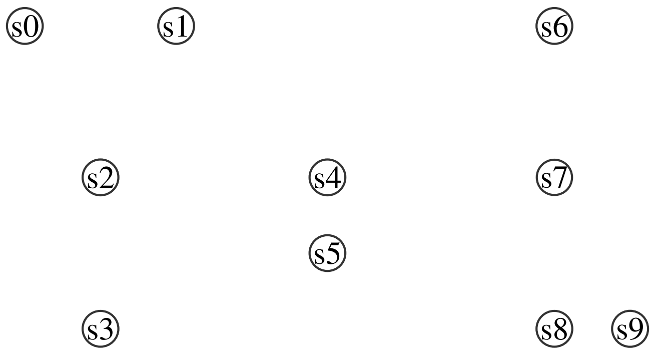
- the set $S_{\text{vis}}(i)$ of the visible leftmost (considering the center of bounding box only) strokes in five directions respectively.

Edges from s_i to the $S_{\text{cro}}(i) \bigcup S_{\text{vis}}(i)$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 ($Crossing, R1, R2, R3, R4, R5$ and $Time$) in the graph. For those edges from s_i to the $S_{\text{cro}}(i) \cap S_{\text{vis}}(i)$, the type $Crossing$ is assigned.

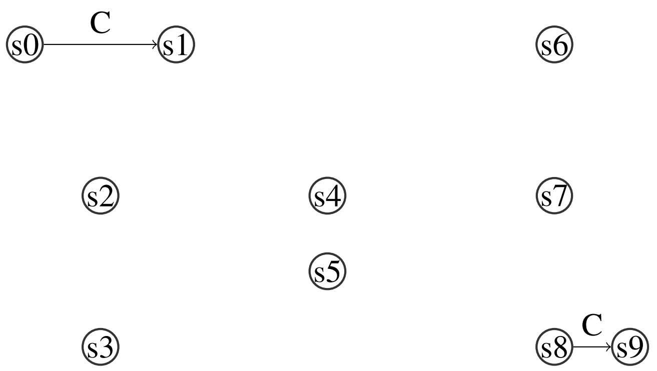
Figure 6.7 illustrates the process of deriving graph from raw input step by step using the example of $\frac{f}{a} = \frac{b}{f}$. First according to 10 strokes in the raw input (Figure 6.7a), we create 10 nodes, one for each stroke (Figure 6.7b); for each stroke, look for its crossing stroke or strokes and add the corresponding edges



(a)

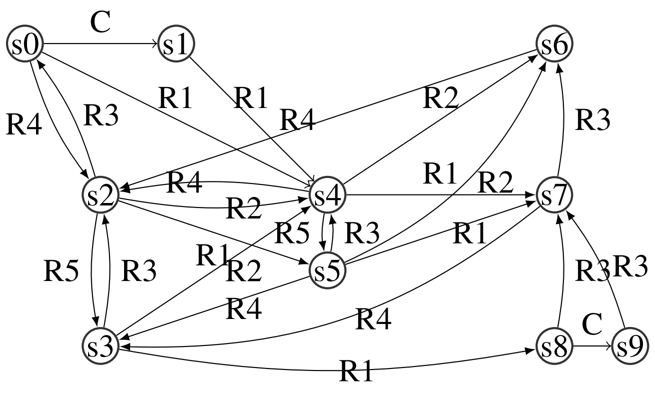


(b)

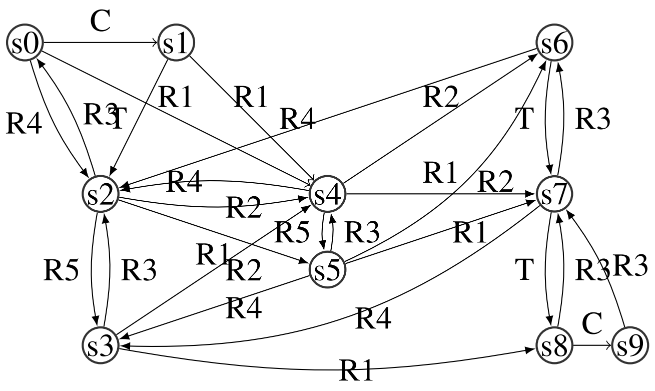


(c)

Figure 6.7 – (a) $\frac{f}{a} = \frac{b}{f}$ is written with 10 strokes; (b) create nodes; (c) add $Crossing$ edges. C : $Crossing$.



(d)



(e)

Figure 6.7 – (d) add $R1, R2, R3, R4, R5$ edges; (e) add $Time$ edges. $C : Crossing, T : Time$.

labeled with $Crossing$ between nodes (Figure 6.7c); 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 (Figure 6.7d); 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 (Figure 6.7e).

6.4.2 Graph evaluation

With the same method adopted in Section 6.4.2 and 4.3.2, we evaluate the new proposed graph representation model. Specifically, provided the ground truth labels of the nodes and edges in the graph, we would like to see evaluation results at symbol and expression levels. We reintroduce the evaluation criteria repeatably here as a kind reminder to readers: symbol segmentation ('Segments'), refers to a symbol that is correctly segmented whatever the label; 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 right relationship label.

Table 6.1 and Table 6.2 present the evaluation results on CROHME 2014 test set (provided the ground truth labels) at the symbol and expression level respectively. We re-show the evaluate results of time graph and the proposed graph in Chapter 5 as a reference to the new graph. Compared to the graph model proposed in Chapter 5, the new graph model stays at the same level with regards to the recall rate and precision rate on symbol segmentation and recognition task. When it comes to relationship classification task, the new graph presents a small improvement, about 0.5%. The new graph catch 93.99% relationships. Owing to the missed 6.00% relationships, around 30% expressions are not correctly recognized as presented in Table 6.2.

Table 6.1 – The symbol level evaluation results on CROHME 2014 test set (provided the ground truth labels).

Model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
time graph	99.73	99.45	99.72	99.44	75.54	99.27
graph (Chapter 5)	100.00	99.99	99.99	99.98	93.48	99.95
new graph	99.97	99.93	99.96	99.92	93.99	99.86

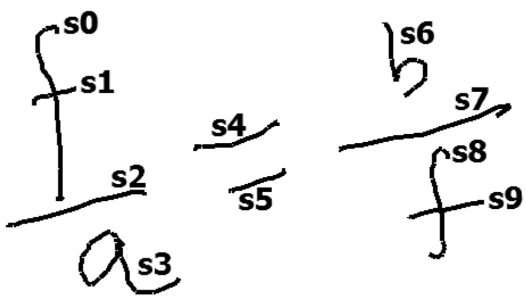
Table 6.2 – The expression level evaluation results on CROHME 2014 test set (provided the ground truth labels).

Model	correct (%)	<= 1 error	<= 2 errors	<= 3 errors
time graph	34.11	40.94	50.51	58.25
graph (Chapter 5)	67.65	76.70	85.76	90.74
new graph	69.89	77.21	85.96	90.54

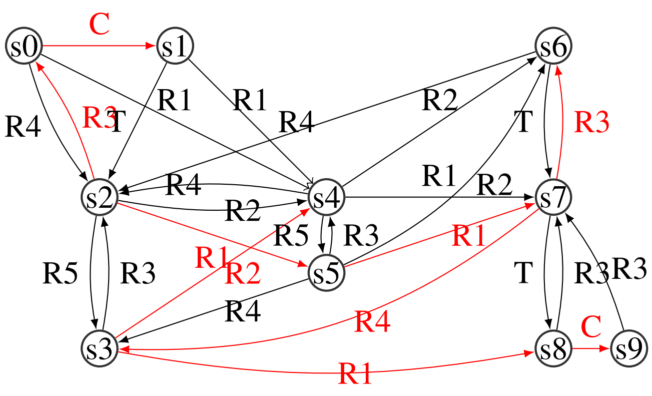
6.4.3 Derivation of trees from G

Heretofore, we derive a graph from the raw input considering the the temporal and spatial information. Figure 6.8 illustrates the ME $\frac{f}{a} = \frac{b}{f}$ written with 10 strokes and the derived graph G. We would like to label nodes and edges of G correctly in order to build a SLG finally. The solution proposed in this chapter is to derive trees from G, then recognize the trees using the tree-based BLSTM model.

There exists 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. For the



(a)



(b)

Figure 6.8 – (a) $\frac{f}{a} = \frac{b}{f}$ is written with 10 strokes; (b) the derived graph G, the red part is one of the possible trees with $s2$ as the root. $C : Crossing, T : Time.$

example illustrated in Figure 6.8a, stroke $s2$ is 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 $(Crossing, R1, R3, R4, R2, R5, Time)$. With this strategy, a tree is derived to which we give the name *Tree-Left-R1* which is dedicated to catch $R1$ relationship. If the visiting order follow $(Crossing, R2, R1, R3, R4, R5, Time)$, another tree named *Tree-Left-R2* would be derived to focus more on $R2$ relationship. Likewise, tree *Tree-Left-R3*, *Tree-Left-R4*, *Tree-Left-R5* are derived respectively to emphasize $R3$, $R4$, $R5$ relationships. The $Crossing$ edge is always on the top of list, and it is because we assume that a pair of crossing strokes belong to a single symbol. In Figure 6.8b, *Tree-Left-R1* is depicted in red with the root in $s2$. Note that in this case, all the nodes are accessible from the start node $s2$. 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 $s0$ since sometimes users start writing an expression from its root. Note that in some cases, the leftmost stroke and stroke $s0$ could be the same one. We replace the left-most stroke with stroke $s0$ and keep the same strategy to derive the trees. These new trees are named as *Tree-0-R1*, Tree-0-R2, Tree-0-R3, Tree-0-R4, Tree-0-R5 respectively.

Finally, if $s0$ 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 $s0 \to s1 \to s2 \to s3 \ldots \to s9$ for the expression in Figure 6.8. Table 6.3 offers an clear look at different types of the derived trees from the graph.

Table 6.3 – The different types of the derived trees.

Type	Root	Traverse algorithm	Visiting order
Tree-Left-R1	the leftmost stroke	Depth-First Search	$(Crossing, R1, R3, R4, R2, R5, Time)$
Tree-Left-R2	the leftmost stroke	Depth-First Search	$(Crossing, R2, R1, R3, R4, R5, Time)$
Tree-Left-R3	the leftmost stroke	Depth-First Search	$(Crossing, R3, R1, R4, R2, R5, Time)$
Tree-Left-R4	the leftmost stroke	Depth-First Search	$(Crossing, R4, R1, R3, R2, R5, Time)$
Tree-Left-R5	the leftmost stroke	Depth-First Search	$(Crossing, R5, R1, R3, R4, R2, Time)$
Tree-0-R1	$s0$	Depth-First Search	$(Crossing, R1, R3, R4, R2, R5, Time)$
Tree-0-R2	$s0$	Depth-First Search	$(Crossing, R2, R1, R3, R4, R5, Time)$
Tree-0-R3	$s0$	Depth-First Search	$(Crossing, R3, R1, R4, R2, R5, Time)$
Tree-0-R4	$s0$	Depth-First Search	$(Crossing, R4, R1, R3, R2, R5, Time)$
Tree-0-R5	$s0$	Depth-First Search	$(Crossing, R5, R1, R3, R4, R2, Time)$
Tree-Time	$s0$	Depth-First Search	only the time order

6.4.4 Feed the inputs of the Tree-based BLSTM

In section 6.4.3, 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 free of the interference of the different writing speed, 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 Figure 6.9, where the re-sampled points are displayed inside the nodes (on-paper points for node) and

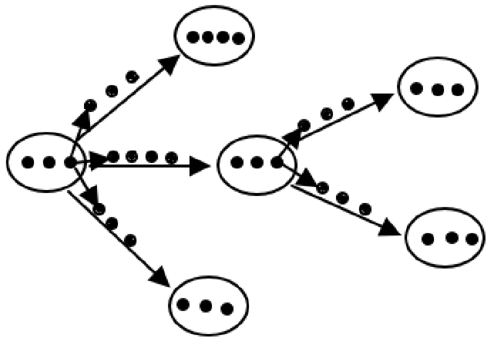


Figure 6.9 – A re-sampled tree. The small arrows between points provide the directions of information flows. With regard to the sequence of points inside one node or edge, most of small arrows are omitted.

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 C, Ri $(i = 1...5)$ 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 size of expression. 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 compute 5 features [sinθ, cosθ, sinϕ, cosϕ, PenUD] which are already described in Section 4.2.2.

6.4.5 Training process

Figure 6.10 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

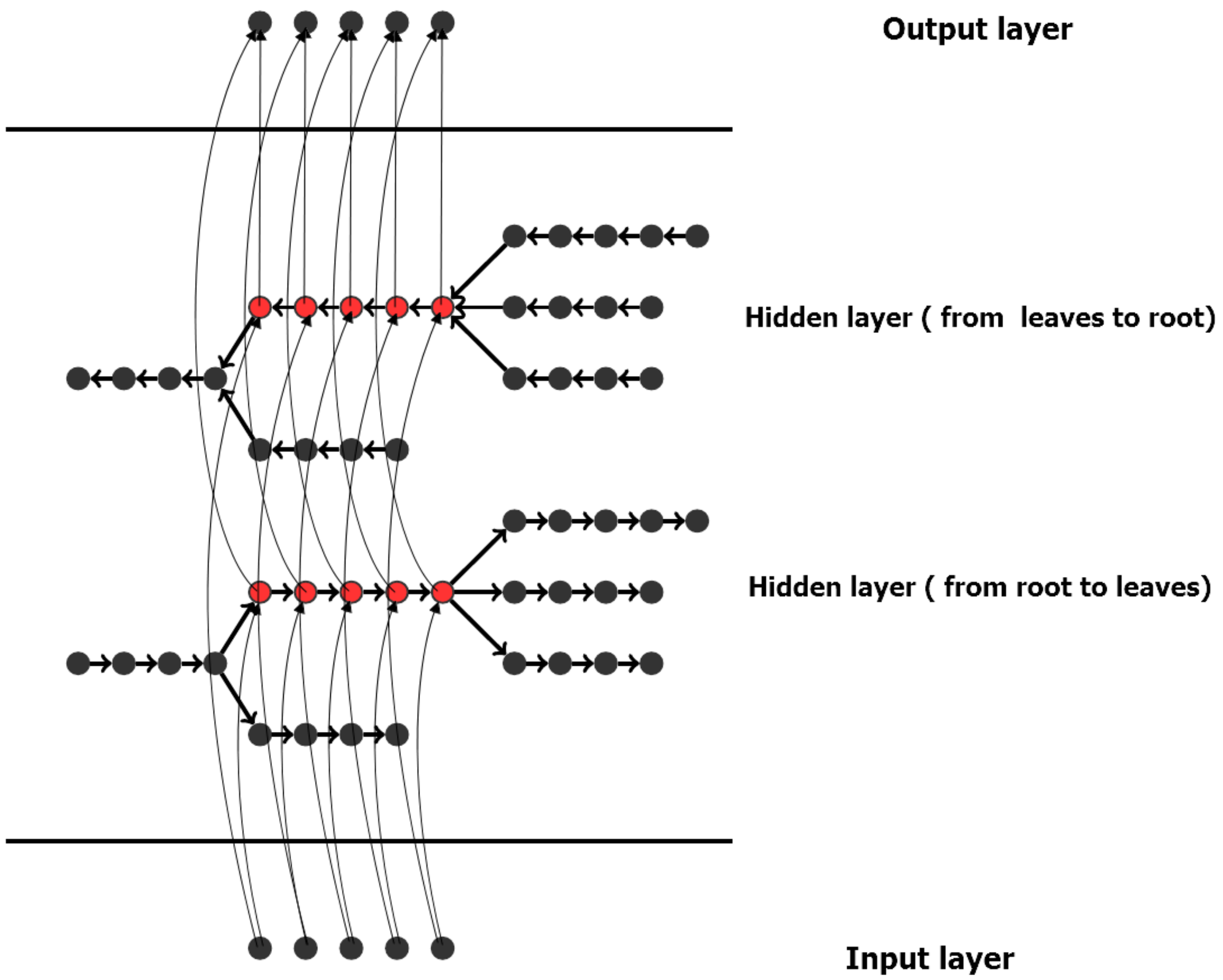

Output layer
Hidden layer (from leaves to root)
Hidden layer (from root to leaves)
Input layer

Figure 6.10 – A tree-based BLSTM network with one hidden level. We only draw the full connection on one short sequence (red) for a clear view.

we are dealing with is a tree; locally, it consists of several short sequences. For example, the tree presented in Figure 6.10 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 Figure 6.10 for example) is similar to the classical BLSTM model except that some outside information 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 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 (Figure 6.11a). 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 (Figure 6.11b). 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 paths of propagation are independent, no pre-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-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 [Bluche et al., 2015], 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. A local CTC algorithm was proposed in Chapter 4 aiming to limit the label into the corresponding stroke at the same time take the advantage of "blank" label, and furthermore it was verified by experiments to outperform the frame wise training method. We succeeded in realizing local CTC for a global sequence labeling task in Chapter 4. In this chapter, we will use local CTC training method in a tree labeling task. With regards to local CTC, the theory behind two types of tasks remain the same actually. The difference between them is: in Chapter 4, a global sequence consisting of several strokes is a entity being processed; here, we treat each short sequence (stroke) as a processing unit.

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 Figure 6.12. The equations provided in Section 4.2.3 are for a global sequence of one or more strokes. In the remaining part of this section, the equations we present are related to a short sequence (a single stroke).

Given the tree input represented as X consisting of N short sequences, each short sequence could be denoted as $X_i, i = 1, ..., N$ 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. Given the above notations, the probability of l_i can be expressed as the sum of the forward variables with and without the final blank at point T_i.

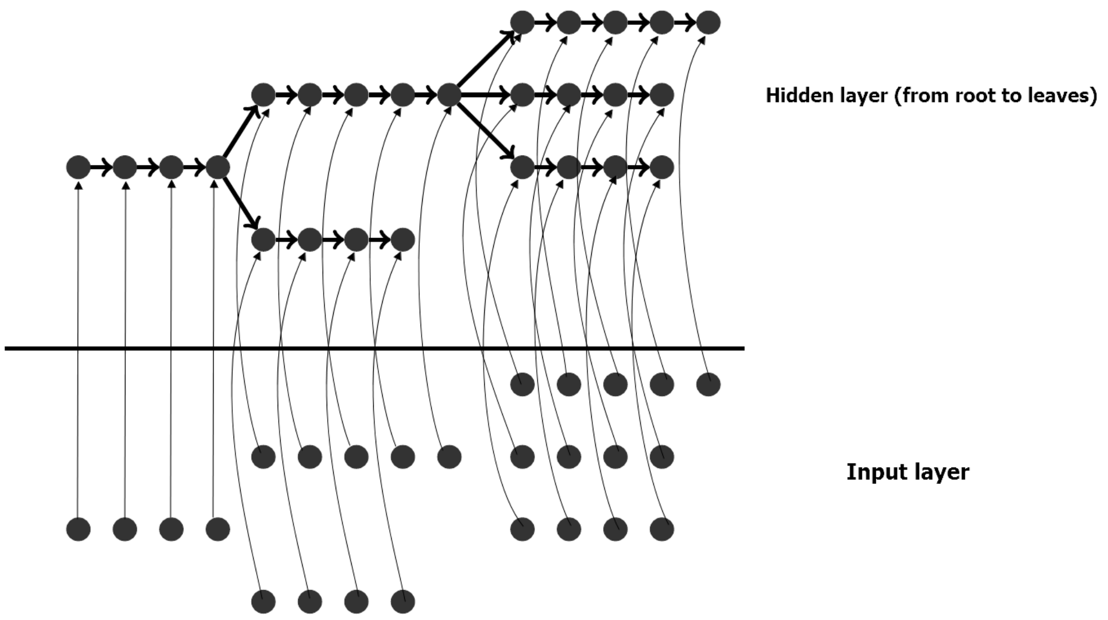
$$p(l_i|X_i) = \alpha_i(T_i, 3) + \alpha_i(T_i, 2) \tag{6.17}$$

$\alpha_i(t, u)$ can be computed recursively as following:

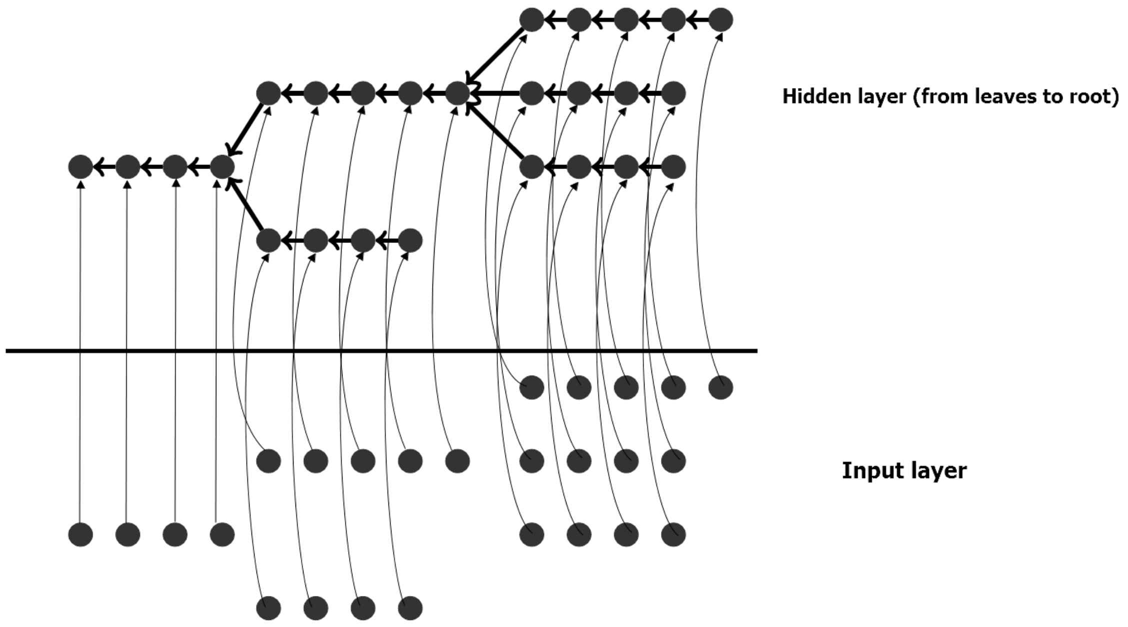
$$\alpha_i(1, 1) = y^1_{blank} \tag{6.18}$$

$$\alpha_i(1, 2) = y^1_{l_i} \tag{6.19}$$

$$\alpha_i(1, 3) = 0 \tag{6.20}$$



(a)



(b)

Figure 6.11 – Illustration for the pre-computation stage of tree-based BLSTM. (a) From the input layer to the hidden layer (from root to leaves), (b) from the input layer to the hidden layer (from leaves to root).

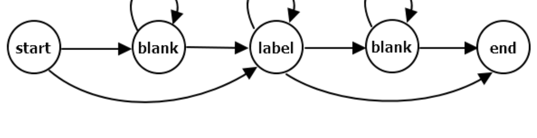


Figure 6.12 – The possible labels of points in one short sequence.

$$\alpha_i(t,u) = y^t_{(l'_i)_u} \sum_{j=u-1}^{u} \alpha_i(t-1,j) \tag{6.21}$$

Note that

$$\alpha_i(T_i,1) = 0 \tag{6.22}$$

$$\alpha_i(t,0) = 0, \forall t \tag{6.23}$$

Figure 6.13 demonstrates the local CTC forward-backward algorithm limited in one stroke.

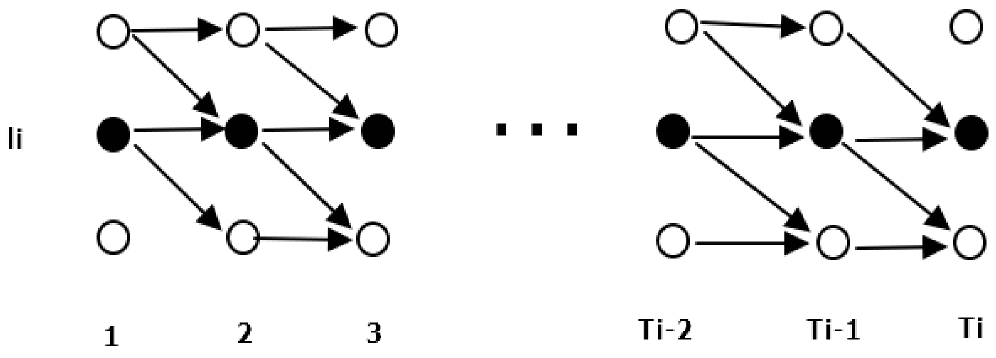


Figure 6.13 – CTC forward-backward algorithm in one stroke X_i. Black circle represents label $l_i$and white circle represents blank. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated in the reverse direction. This figure is a local part (limited in one stroke) of Figure 4.8.

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 $\alpha_i(t,u)$. The formulas for the initialization and recursion of the backward variable are as follows:

$$\beta_i(T_i,3) = 1 \tag{6.24}$$

$$\beta_i(T_i,2) = 1 \tag{6.25}$$

$$\beta_i(T_i,1) = 0 \tag{6.26}$$

$$\beta_i(t,u) = \sum_{j=u}^{u+1} \beta_i(t+1,j) y^{t+1}_{(l'_i)_j} \tag{6.27}$$

Note that

$$\beta_i(1,3) = 0 \tag{6.28}$$

$$\beta_i(t,4) = 0, \forall t \tag{6.29}$$

With the local CTC forward-backward algorithm, 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 CTC loss function $L(X_i,l_i)$ is defined as the negative log probability of correctly labeling the short sequence X_i:

$$L(X_i,l_i) = -\ln p(l_i|X_i) \tag{6.30}$$

According to the Equation 3.48, we can rewrite $L(X_i,l_i)$ as:

$$L(X_i,l_i) = -\ln \sum_{u=1}^{3} \alpha_i(t,u)\beta_i(t,u) \tag{6.31}$$

Then the errors will be back propagated to the output layer (Equation 3.49), the hidden layer (Equation 3.50), 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:

$$L(X,l) = \sum_{i=1}^{N} L(X_i,l_i) \tag{6.32}$$

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.

6.4.6 Recognition process

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

6.4.7 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 which are not used when building the SLG in previous chapter. 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 directions are both one of $(Right, Above, Below, Inside, Superscript, Subscript)$ as illustrated in Figure 6.14a, we also choose the one having the larger probability. (b) Another type of conflict could be the case illustrated in Figure 6.14b where one symbol has two (or more) input relationships (one of 6 relationships). When observing the structure of SRTs, we can easily find that 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 should be a connected tree (this is a structural constraint, not a language specific constraint), there exist one root node and one or multiple leaf nodes inside each SRT. Each node has

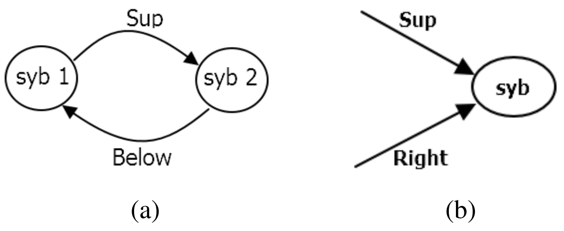


Figure 6.14 – Possible relationship conflicts existing in merging results.

only one input edge, except the root node. After performing the first three steps, we still have the probability to output a SRT consisting 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.

6.5 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. 1147 expressions are included in CROHME 2016 test data set.

Setup. We constructed the tree-based BLSTM recognition system with the RNNLIB library[1]. As described in Section 3.3.4, DBLSTM [Graves et al., 2013] can be created by stacking multiple BLSTM layers on top of each other in order to get higher level representation of the input data. Several types of configurations are included in this chapter: *Networks (i)*, (ii), (iii) and (iv). 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 [Graves et al., 2009] and handwritten math symbol classification [Álvaro et al., 2013, 2014a]. *Network (ii)* is a deep structure with two bidirectional hidden levels, each containing two opposite LSTM layers of 100 cells. *Network (iii)* and *Network (iv)* 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 + $NoRelation$ + $blank$).

Evaluation. With the Label Graph Evaluation library (LgEval) [Mouchère et al., 2014], 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.

6.5.1 Experiment 1

In this experiment, we would like to see the effects of the depth of the network on the recognition results. Then according to the results, we choose the proper network configurations for the task. For each expression, only *Tree-Time* is derived to train the classifier.

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

The evaluation results on symbol level and global expression level are presented in Table 6.4 and 6.5 respectively. From the tables, we can conclude that as the network turns to be deeper, the recognition

Table 6.4 – The symbol level evaluation results on CROHME 2014 test set with *Tree-Time* only.

Network, model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
i, *Tree-Time*	92.93	84.82	84.12	76.78	60.70	76.19
ii, *Tree-Time*	95.10	90.47	87.53	83.27	65.06	83.18
iii, *Tree-Time*	95.43	91.13	88.26	84.28	65.45	83.57
iv, *Tree-Time*	95.57	91.21	87.81	83.80	65.98	82.85

Table 6.5 – The expression level evaluation results on CROHME 2014 test set with *Tree-Time* only.

Network, model	correct (%)	$\leq$ 1 error	$\leq$ 2 errors	$\leq$ 3 errors
i, *Tree-Time*	12.41	20.24	26.14	30.93
ii, *Tree-Time*	16.09	25.46	32.28	37.27
iii, *Tree-Time*	16.80	25.56	32.89	38.09
iv, *Tree-Time*	16.19	25.97	33.20	38.09

rate first increases and then stays at a relatively stable level. There is a large increase from *Network (i)* to *Network (ii)*, a slight increase from *Network (ii)* to *Network (iii)* and no improvement from *Network (iii)* to *Network (iv)*. These results show that 3 bidirectional hidden levels in the network is a proper option for the task in this thesis. A network with depth larger than 3 brings no improvement but higher computational complexity. Thus, for the coming experiments we will not take into account *Network (iv)* any more.

6.5.2 Experiment 2

In this section, we carry out the experiments by merging several trees. As a first try, we derive only 3 trees , *Tree-Time*, *Tree-Left-R1* and *Tree-0-R1* for each expression to train classifiers separately. With regards to each tree, we consider 3 network configurations, being *Network (i)*, *Network (ii)*, *Network (iii)*. 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 (i, *Merge3*), (ii, *Merge3*), (iii, *Merge3*). Then we merge the 3 trees labeled by all these 9 classifiers to obtain the system *Merge9*.

The evaluation results on symbol level and global expression level are presented in Table 6.6 and 6.7 respectively. We give both the individual tree recognition results and the merging results 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 (iii, *Tree-Time*), the symbol segmentation and classification results of (iii, *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 (ii) are systematically increased when compared to (i) as the deep structure could get higher level representations of the input data. The performance of network (iii) is moderately improved when compared to (ii), 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 result 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

Table 6.6 – The symbol level evaluation results on CROHME 2014 test set with 3 trees, along with CROHME 2014 participant results.

Network, model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
i, *Tree-Time*	92.93	84.82	84.12	76.78	60.70	76.19
i, *Tree-Left-R1*	84.82	72.49	72.80	62.21	44.34	57.78
i, *Tree-0-R1*	85.31	72.88	74.17	63.37	42.92	60.08
i, *Merge3*	**93.53**	**87.20**	**86.10**	**80.28**	**71.16**	**66.13**
ii, *Tree-Time*	95.10	90.47	87.53	83.27	65.06	83.18
ii, *Tree-Left-R1*	86.71	75.64	76.85	67.03	48.14	61.91
ii, *Tree-0-R1*	87.52	76.66	77.00	67.45	48.14	63.04
ii, *Merge3*	**95.01**	**90.05**	**88.38**	**83.76**	**76.20**	**72.28**
iii, *Tree-Time*	95.43	91.13	88.26	84.28	65.45	83.57
iii, *Tree-Left-R1*	88.03	78.13	78.56	69.72	50.31	65.87
iii, *Tree-0-R1*	87.41	77.02	77.63	68.40	48.23	64.28
iii, *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**
system	CROHME 2014 participant results					
III	98.42	98.13	93.91	93.63	94.26	94.01
I	93.31	90.72	86.59	84.18	84.23	81.96
VII	89.43	86.13	76.53	73.71	71.77	71.65
V	88.23	84.20	78.45	74.87	61.38	72.70
IV	85.52	86.09	76.64	77.15	70.78	71.51
VI	83.05	85.36	69.72	71.66	66.83	74.81
II	76.63	80.28	66.97	70.16	60.31	63.74

Table 6.7 – The expression level evaluation results on CROHME 2014 test set with 3 trees, along with CROHME 2014 participant results.

Network, model	correct (%)	$\leq$ 1 error	$\leq$ 2 errors	$\leq$ 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**
system	CROHME 2014 participant results			
III	62.68	72.31	75.15	76.88
I	37.22	44.22	47.26	50.20
VII	26.06	33.87	38.54	39.96
VI	25.66	33.16	35.90	37.32
IV	18.97	28.19	32.35	33.37
V	18.97	26.37	30.83	32.96
II	15.01	22.31	26.57	27.69

all the participated systems. When we compute the recognition rate with ≤ 3 errors, our result is 50.15%, very close to the second-ranked system (50.20%).

The top ranked system is from My Script company and they use a much larger training data set which is not available to the public. Furthermore, as we know, 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.

To have more uptodate comparisons, we also evaluate the system of *Merge 9* on CROHME 2016 test data set. As can be seen in Table 6.8, 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 is presented in Table 6.9. The global expression recognition rate is 27.03%.

Table 6.8 – 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 6.9 – 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

6.5.3 Experiment 3

In Experiment 2, we consider merging 3 trees to build a SLG describing an math expression. Compared to the results of symbol segmentation and recognition, the relationship classification results are not particularly prominent. We thought one of the possible reasons could be that only 3 trees can not well cover the graph G, in other words, some edges in the graph G are not used in the already derived 3 trees. Thus, in this experiment, we will test all the 11 trees illustrated in Table 6.3 to see if more trees could improve the results of relationship classification task. Taking into consideration that we see a small increase in recognition results, but a much increase in time complexity from *Network (ii)* to *Network (iii)* in previous experiments, we use *Network (ii)* as a cost-effective choice to train 11 different classifiers in this section.

The evaluation results on symbol level and global expression level are presented in Table 6.10 and 6.11 respectively. In each table, we provide in detail the individual tree recognition results and the merging

Table 6.10 – The symbol level evaluation results on CROHME 2014 test set with 11 trees.

Network, model	Segments (%)		Seg + Class (%)		Tree Rels. (%)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
ii, *Tree-Time*	95.10	90.47	87.53	83.27	65.06	83.18
ii, *Tree-Left-R1*	86.71	75.64	76.85	67.03	48.14	61.91
ii, *Tree-0-R1*	87.52	76.66	77.00	67.45	48.14	63.04
ii, *Tree-Left-R2*	87.09	76.43	75.87	66.59	46.85	60.94
ii, *Tree-0-R2*	88.71	78.80	76.93	68.33	48.48	65.54
ii, *Tree-Left-R3*	86.49	75.54	75.11	65.60	45.19	59.76
ii, *Tree-0-R3*	86.97	75.85	75.68	66.01	45.31	59.69
ii, *Tree-Left-R4*	87.61	77.36	77.93	68.81	49.51	64.91
ii, *Tree-0-R4*	88.88	78.95	78.93	70.11	49.00	65.98
ii, *Tree-Left-R5*	87.47	77.09	77.5	68.31	47.82	63.94
ii, *Tree-0-R5*	88.53	78.66	79.09	70.28	48.27	65.22
ii, *Merge3*	**95.01**	**90.05**	**88.38**	**83.76**	**76.20**	**72.28**
ii, *Merge11*	**94.53**	**89.33**	**87.97**	**83.13**	**77.15**	**72.85**

Table 6.11 – The expression level evaluation results on CROHME 2014 test set with 11 trees.

Network, model	correct (%)	$\leq$ 1 error	$\leq$ 2 errors	$\leq$ 3 errors
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, *Tree-Left-R2*	8.04	13.22	17.19	21.16
ii, *Tree-0-R2*	7.12	14.04	18.31	24.72
ii, *Tree-Left-R3*	6.71	11.50	16.38	21.16
ii, *Tree-0-R3*	6.21	13.02	16.99	21.26
ii, *Tree-Left-R4*	8.55	15.06	19.43	23.19
ii, *Tree-0-R4*	8.95	14.55	20.55	26.04
ii, *Tree-Left-R5*	8.04	14.55	20.35	24.31
ii, *Tree-0-R5*	8.65	15.26	20.24	24.62
ii, *Merge3*	**25.94**	**36.72**	**42.32**	**46.59**
ii, *Merge11*	**25.94**	**37.23**	**42.32**	**45.68**

results (*Merge3* and *Merge11*). As can be seen, all trees have the similar recognition results except *Tree-Time*. And more trees do bring some effects on relationship classification task. Compared to (ii, *Merge3*), the results of relationship classification are slightly improved around 1% while symbol segmentation and recognition results are slightly reduced around 0.5%. Finally, at expression level, we see no significant changes as shown in Table 6.11.

6.6 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) [Mouchère et al., 2014] evaluate 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 to us with the library. Based on the two confusion matrices, we analyze the errors specifically below.

Node label In table 6.12, we list the types of SLG node label error which has 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; the last column records the corresponding no. of occurrences. As can be seen from the table, the most frequent error ($x \rightarrow X$, 46) belongs to the type of the lowercase-uppercase error. 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 error. 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. Theoretically, two main types of node label error, being the lowercase-uppercase error and the similar-look error, could be eased when more training data is introduced. Thus, one of future work could be trying to collect new data as much as possible.

Table 6.12 – Illustration of node (SLG) label errors of (*Merge 9*) on CROHME 2014 test set. We only list the cases that occur $\geq$ 10 times.

output label	ground truth label	no. of occurrences
x	X	46
x	$\times$	26
p	P	24
,	1	19
c	C	16
y	Y	14
+	t	14
.	...	13
X	x	16
a	x	14
1	|	11
-	1	10
$\times$	x	10
z	2	10
q	9	10

Edge label Table 6.13 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, maybe the classifiers answer too often this frequent class.

Now we will explore deeper the problem of the missing edges which appear in the last row of the Table 6.13. In fact, there exist three sources which result in the missing edges: (1) the edges are missed during the

Table 6.13 – Illustration of edge (SLG) label errors of (*Merge 9*) on CROHME 2014 test set. The first column represents the output labels; the first row offers the ground truth labels; 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 a overall label '*'.).

	*	Above	Below	Inside	Right	Sub	Sup	_
*	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	

graph representation stage. We evaluated the graph model in Section 6.4.2 where around 6% relationships were missed. One of the future works could be searching for a better graph representation model to catch the 6% 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) When deriving multiple trees from the graph G, they do not well cover the graph completely. We have tried to ease the source 3 by deriving more trees, for example 11 trees in Experiment 3. However, the idea of using more trees did not work well in fact. Thus, a better strategy for deriving trees from the graph will be explored in future works.

We reconsider the 2 test samples ($a \geq b$ and $44 - \frac{4}{4}$) from CROHME 2014 test set recognized by system (*Merge9*). We provide the handwritten input, the derived graph from the raw input, the derived trees from the graph, along with the built SLG for each test sample (Figure 6.15 and Figure 6.16). These 2 samples were recognized correctly by system (*Merge9*). As shown, several extra edges appear owing to multiple trees and they were all recognized correctly as $NoRelation$. We remove these $NoRelation$ edges to have a intuitive judgment on the recognition result (Figure 6.15f and 6.16f).

In addition, we present a failed case in Figure 6.17. Just like the previous samples, we illustrate the handwritten input of $\frac{9}{9+\sqrt{9}}$, the derived graph from the raw input, the derived trees from the graph, as well as the final SLG built. As can be seen, the structure of this expression 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 language model since $\frac{9}{\rightarrow+\sqrt{9}}$ is not a valid expression from the point of language model.

6.7 Discussion

In this chapter, we extended the classical BLSTM to tree-based BLSTM and applied the new network model for recognizing online mathematical expressions. The new model has a tree topology, and possesses the ability to model directly the dependency among the tree-structured data. The proposed tree-based

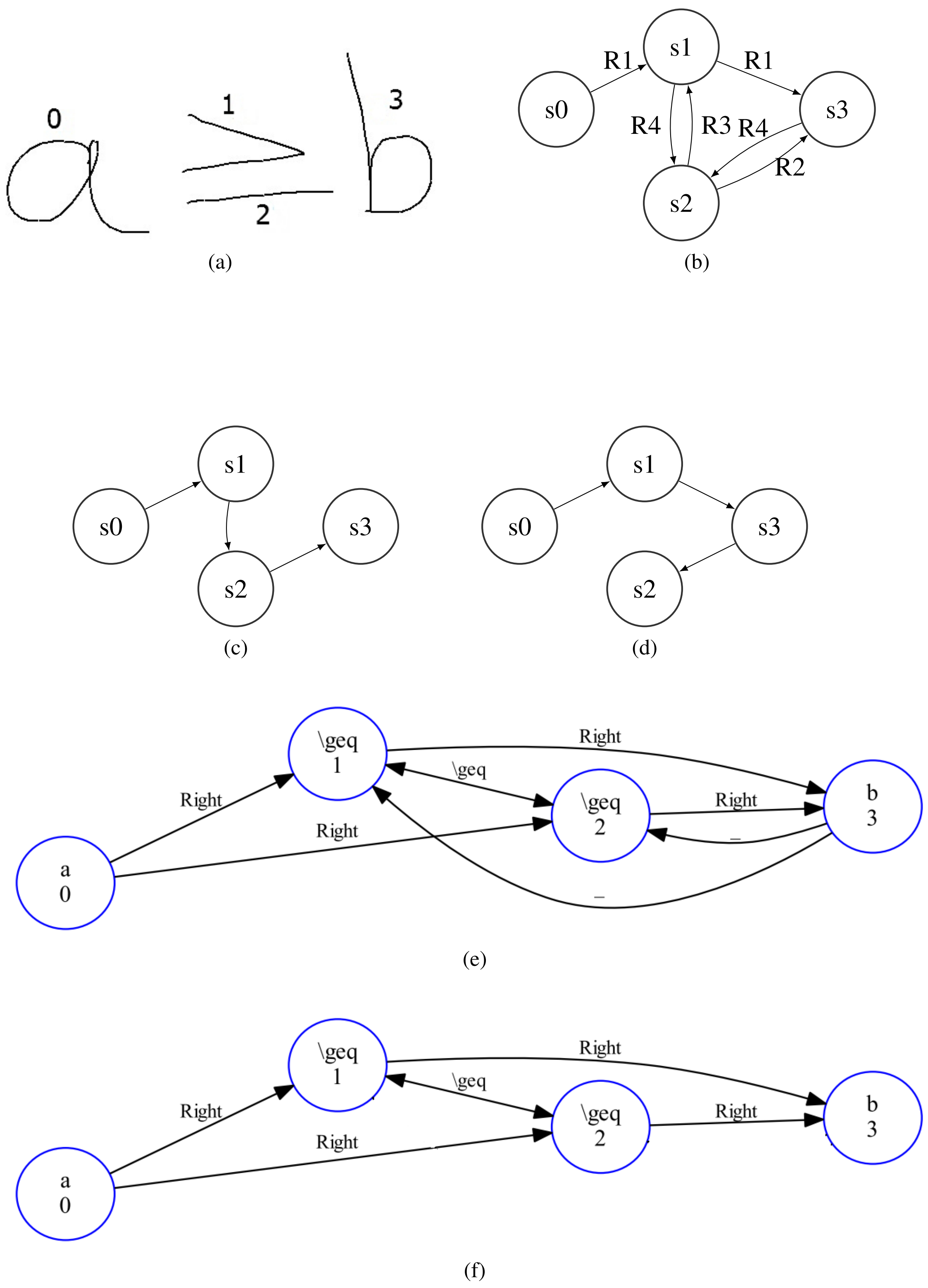
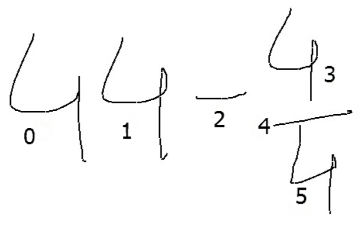
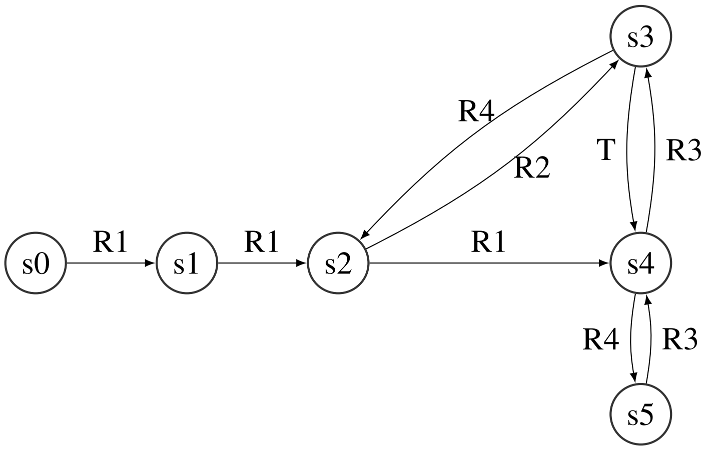


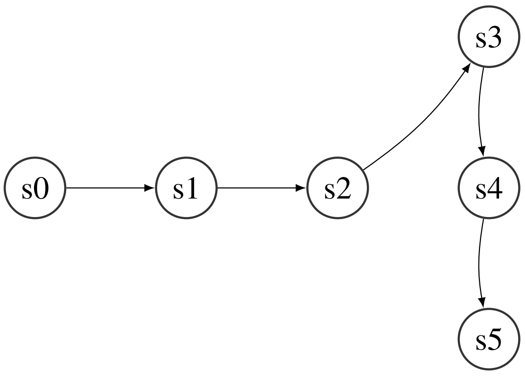
Figure 6.15 – (a) $a \geq b$ written with four strokes; (b) the derived graph; (b) *Tree-Time*; (c)*Tree-Left-R1* (In this case, *Tree-0-R1* is the same as *Tree-Left-R1*); (e) the built SLG of $a \geq b$ after merging several trees and performing other post process steps, all labels are correct; (f) the built SLG with $NoRelation$ edges removed.



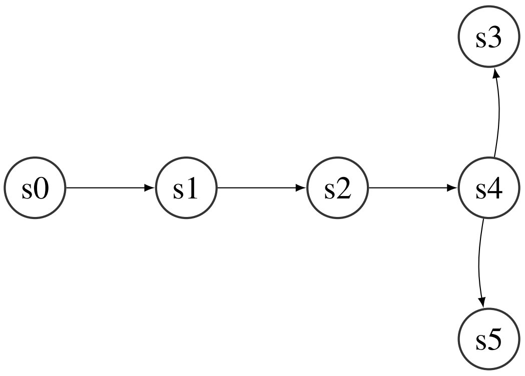
(a)



(b)

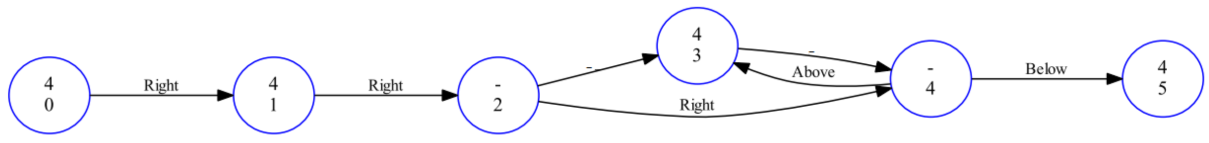


(c)

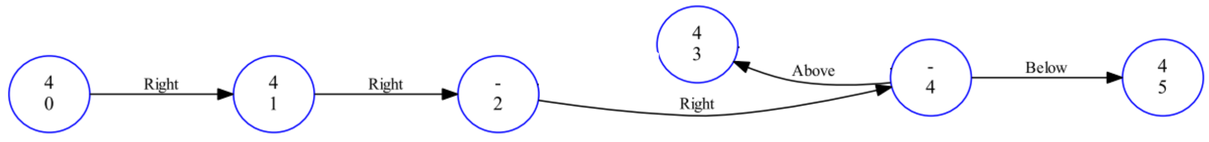


(d)

Figure 6.16 – (a) $44 - \frac{4}{4}$ written with six strokes; (b) the derived graph; (b) *Tree-Time*; (c)*Tree-Left-R1* (In this case, *Tree-0-R1* is the same as *Tree-Left-R1*);



(e)

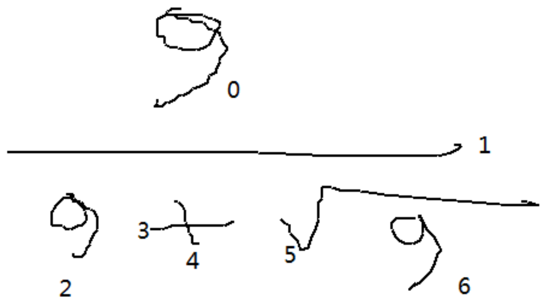


(f)

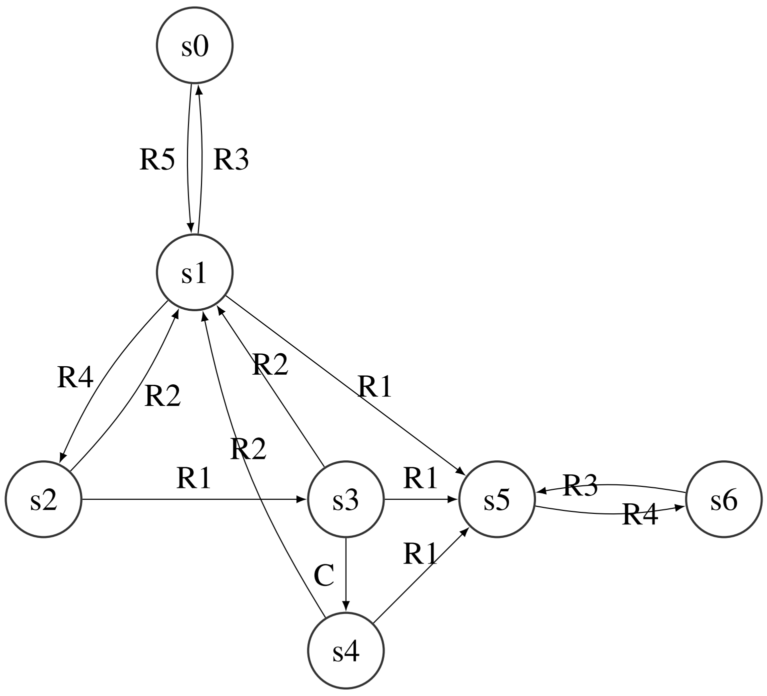
Figure 6.16 – (b)the built SLG after merging several trees and performing other post process steps; (c) the built SLG with $NoRelation$ edges removed.

BLSTM system, requiring no high time complexity or manual work involved in the classical grammar-driven systems, achieves competitive results in online mathematical expression recognition domain. Another 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. With regard to the symbol segmentation and classification, the proposed system performs better than the second-ranked system in CROHME 2014 (the top ranked system used a much larger training data set which is not available to the public). For relationship recognition, we achieve better results than the third-ranked system. When considering the expression recognition rate with ≤ 3 errors, our result is 50.15%, close to the second-ranked system (50.20%).

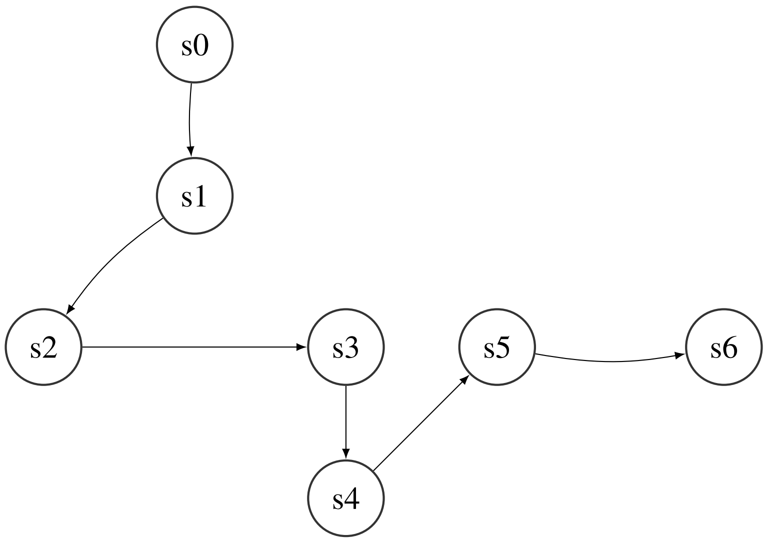
In future, several directions could be explored to extend the current work. (1) As we analyzed in Section 6.6, we could put efforts into collecting more training data to ease the lowercase-uppercase error and the similar-look error. (2) The current graph model misses still around 6% relationships on CROHME 2014 test set. Now, only one rule is used to define the visibility between a pair of strokes. In future, we will try to set several rules for defining the visibility between strokes. As long as a pair of strokes meet any one among these several rules, we decide that they could see each other. (3) A better strategy for deriving trees from the graph should be explored to get a better coverage of the graph. (4) As we cover more and more edges, the precision rate will decrease relevantly as presented in the previous experiments. Thus, one future direction could be developing a better training protocol to enforce the training of the class $NoRelation$. Then, a stronger post process step should be considered to improve the recognition rate.



(a)

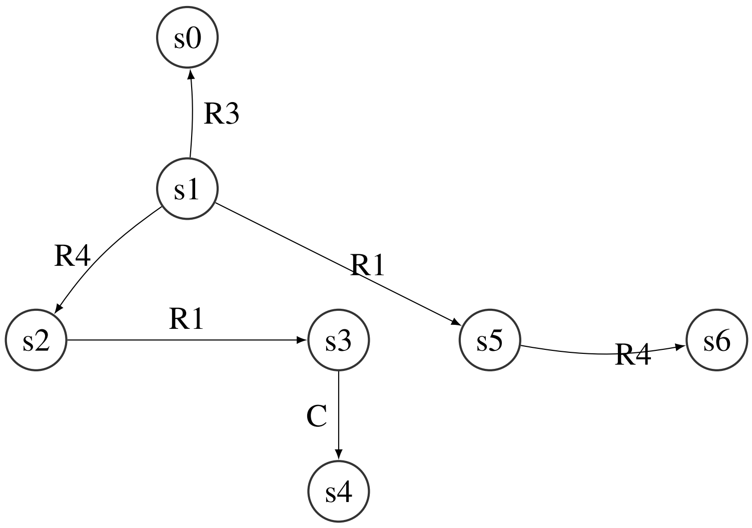


(b)

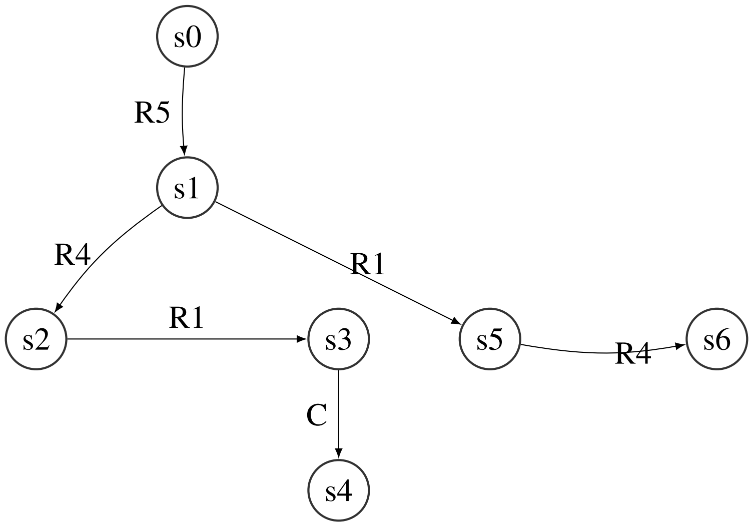


(c)

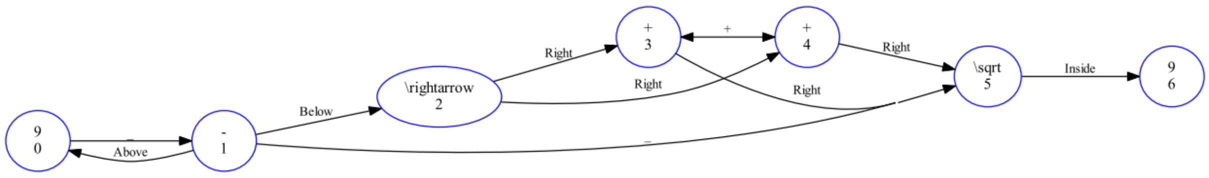
Figure 6.17 – (a) $\frac{9}{9+\sqrt{9}}$ written with 7 strokes; (b) the derived graph; (b) *Tree-Time*;



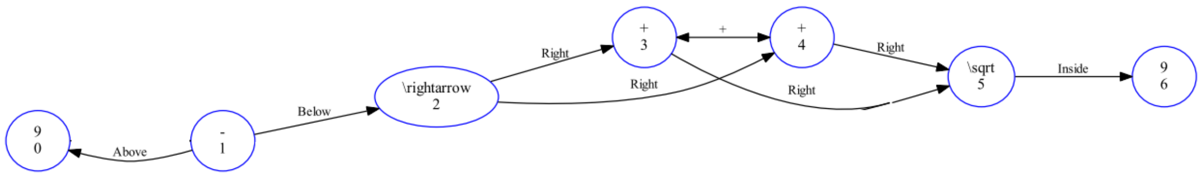
(d)



(e)



(f)



(g)

Figure 6.17 – (d)*Tree-Left-R1*; (e)*Tree-0-R1*; (f)the built SLG after merging several trees and performing other post process steps; (g) the built SLG with $NoRelation$ edges removed. There is a node label error: the stroke 2 with the ground truth label '9' was wrongly classified as '$\rightarrow$'.

7 Conclusion and future works

In this chapter, we first summarize the works of the thesis and list the main contributions made during the research process. Then, based on the current method and experiments results, we will propose several possible directions for future work.

7.1 Conclusion

We study the problem of online mathematical expression recognition in this thesis. Generally, ME recognition involves three tasks: symbol segmentation, symbol recognition and structural analysis [Zanibbi and Blostein, 2012]. The state of the art solutions, considering the natural relationship between the three tasks, perform these 3 tasks at the same time by using grammar parsing techniques. Commonly, a complete grammar for math expressions consists of hundreds of production rules. These rules need to be designed manually and carefully for different data sets. Furthermore, the time complexity for grammar-driven parsing is usually exponential if no constraints are set to control it. Thus, to bypass the high time complexity and manual work of the classical grammar-driven systems, we proposed a new architecture for online mathematical expression recognition in this thesis. The backbone of our system is the framework of BLSTM recurrent networks with a CTC output layer, which achieved great success in sequence labeling task such as text and speech recognition thanks to the ability of learning long-term dependency and the efficient training algorithm.

Mathematical expression recognition with a single path. Since BLSTM network with a CTC output layer is capable of processing sequence-structured data, as a first step to try, we proposed a trivial strategy where a BLSTM directly labelled the sequence of pen-down and pen-up strokes respecting the time order. These later added strokes (pen-up strokes) are used to represent the relationships between pairs of visible strokes by assigning them a ground truth label. In order to assign each stroke (visible or later added) a label in the recognition process, we extended the CTC training technique to local CTC, constraining the output labels into the corresponding strokes and at the same time benefiting from introducing an addition ’blank’ class. At last, we built the 2-D expression from the outputted sequence of labels. The main contributions in this first proposal consist of: (1) We propose a new method to represent the relationship of a pair of visible strokes by linking the last point and the first point of them. With this method, a global sequence is generated and could be coped with BLSTM and CTC topology. (2) We extend the CTC training technique to local CTC. The new training technique proposed could improve the system performance globally compared to frame-wise training, as well constrain the output relatively. The limitation of this simple proposal is that it takes into account only a pair of visible strokes successive in the input time, and therefore miss some relationships for 2-D mathematical expressions.

Mathematical expression recognition by merging multiple paths. In the above-mentioned simple proposal, we considered only the pairs of strokes which are successive in the time order. Obviously, a sequence-structured model is not able to cover all the relationships in 2-D expressions. Thus, we turned to a graph structure to model the relationships between strokes in mathematical expressions. Globally, the input of the recognition system is an handwritten expression which is a sequence of strokes; the output is the stroke label graph which consists of the information about the label of each stroke and the relationships between stroke pairs. Firstly, we derived an intermediate graph from the raw input using both the temporal and spatial information between strokes. In this intermediate graph, each node represents a stroke and edges are added according to temporal or spatial properties between strokes, which represent the relations of stroke pairs. Secondly, several 1-D paths were selected from the graph since the classifier model used is a 1-D sequence labeler. Next, we used the BLSTM classifier to label the selected 1-D paths. Finally, we merged these labeled paths to build a complete stroke label graph. Compared to the proposal with a single path, the solution by merging multiple paths presented improvements on recall rate of 'Tree Rels.' but at the same time decreases the precision rate of 'Tree Rels.' Thus, at the expression level, the recognition rate remained the same level as the solution with single path.

One main contribution of this proposal is that multiple paths are used to represent a 2-D expression. However, even though several paths from one expression were considered in this system, the BLSTM model dealt with each path separately in essential. The classical BLSTM model could access information from past and future in a long range but the information outside the single sequence is of course not accessible to it. In fact, it is not the real case where human beings recognize the raw input using the entire contextual information.

Mathematical expression recognition by merging multiple trees. As explained above, human beings interpret handwritten math expression by considering the global contextual information. In the system by merging multiple paths, each path was processed separately implying that only contextual information in the path could be visited. Thus, we developed a neural network model which could handle directly a structure not limited to a chain. We extended the chain-structured BLSTM to tree structure topology and applied this new network model for online math expression recognition. With this new neural network model, we could take into account the information in a tree instead of a single path at one time when dealing with one expression. Similar to the framework of the solution by merging multiple paths, we first derived an intermediate graph from the raw input. Then, instead of 1-D paths, we considered from the graph deriving trees which would be labeled by tree-based BLSTM model as a next step. In the end, these labeled trees were merged to build a stroke label graph. Compared to the proposal by merging multiple paths, the new recognition system was globally improved which was verified by experiments. One main contribution of this part is that we extend the chain-structured BLSTM to tree-based BLSTM, and provide the new topology with the ability of modeling dependency in a tree.

We list the main **contributions** here:

- 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.
- We propose a new method to represent the relationship of a pair of visible strokes by linking the last point and the first point of them.
- We extend the CTC training technique to local CTC. The new training technique proposed could improve the system performance globally compared to frame-wise training, as well constrain the output relatively.
- We extend the chain-structured BLSTM to tree-based BLSTM, and provide the new topology with the ability of modeling dependency in a tree.
- The proposed system, without using any grammar, achieves competitive results in online math expression recognition domain.

7.2 Future works

Based on the current method and error analysis, we summarize here 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 for us. Actually a master project have been proposed already in this direction.
- 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. So we could leave the stage of deriving trees aside.
- 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 thesis, 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 algorithm are generic ones and we will apply them into other research fields in future.

Bibliography

F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1012–1016. IEEE, 2013. 66, 71, 89, 111

F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. Offline features for classifying handwritten math symbols with recurrent neural networks. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 2944–2949. IEEE, 2014a. 71, 89, 111

F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 35: 58–67, 2014b. 19, 34

F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. An integrated grammar-based approach for mathematical expression recognition. *Pattern Recognition*, 51:135–147, 2016. 9, 19, 34, 37, 39, 42

W. Aly, S. Uchida, A. Fujiyoshi, and M. Suzuki. Statistical classification of spatial relationships among mathematical symbols. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 1350–1354. IEEE, 2009. 34

R. H. Anderson. Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium*, pages 436–459. ACM, 1967. 16, 34

R. H. Anderson. Two-dimensional mathematical notation. In *Syntactic Pattern Recognition, Applications*, pages 147–177. Springer, 1977. 16, 34

A.-M. Awal, H. Mouchère, and C. Viard-Gaudin. A global learning approach for an online handwritten mathematical expression recognition system. *Pattern Recognition Letters*, 35:68–77, 2014. 9, 19, 34, 36, 66

P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11):937–946, 1999. 47

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. 48

C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995. 44

D. Blostein and A. Grbavec. Recognition of mathematical notation. *Handbook of character recognition and document image analysis*, 21:557–582, 1997. 16, 34

T. Bluche, H. Ney, J. Louradour, and C. Kermorvant. Framewise and ctc training of neural networks for handwriting recognition. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 81–85. IEEE, 2015. 66, 107

T. Bluche, J. Louradour, and R. Messina. Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention. *arXiv preprint arXiv:1604.03286*, 2016. 95

H. A. Bourlard and N. Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media, 2012. 53

M. Celik and B. Yanikoglu. Probabilistic mathematical formula recognition using a 2d context-free graph grammar. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 161–166. IEEE, 2011. 19, 34

K.-F. Chan and D.-Y. Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, 2000. 16, 34

S.-K. Chang. A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, 2(3):253–272, 1970. 16, 34

P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. *Visual Communications and Image Processing IV*, 1199:852–863, 1989. 17, 34

T. H. Cormen. *Introduction to algorithms*. MIT press, 2009. 80

J. H. Davenport and M. Kohlhase. Unifying math ontologies: A tale of two standards. In *Calculemus/MKM*, pages 263–278. Springer, 2009. 29

M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry: Algorithms and applications. 80, 81

Y. Deng, A. Kanervisto, and A. M. Rush. What you get is what you see: A visual markup decompiler. *arXiv preprint arXiv:1609.04938*, 2016. 10, 34, 39, 41

M. Dewar. Openmath: an overview. *ACM SIGSAM Bulletin*, 34(2):2–5, 2000. 29

I. C. Giles, S. Hanson, and J. Cowan. Holographic recurrent networks. 48

A. Graves. Rnnlib: A recurrent neural network library for sequence learning problems, 2013.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 4, pages 2047–2052. IEEE, 2005. 51

A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009. 95

A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009. 71, 89, 111

A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013. 51, 111

A. Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012. 10, 43, 44, 46, 48, 49, 51, 53, 54, 55, 56, 57, 65, 69, 95, 97

N. S. Hirata and W. Y. Honda. Automatic labeling of handwritten mathematical symbols via expression matching. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 295–304. Springer, 2011. 10, 80

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 48

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. 48

L. Hu. *Features and Algorithms for Visual Parsing of Handwritten Mathematical Expressions*. PhD thesis, Rochester Institute of Technology, 2016. 10, 79, 80, 81, 84

L. Hu and R. Zanibbi. Segmenting handwritten math symbols using adaboost and multi-scale shape context features. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1180–1184. IEEE, 2013. 80

A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996. 44

F. Julca-Aguilar. *Recognition of Online Handwritten Mathematical Expressions using Contextual Information*. PhD thesis, Université de Nantes; Université Bretagne Loire; Universidade de São Paulo, 2016. 10, 19, 34, 37, 40

M. Koschinski, H.-J. Winkler, and M. Lang. Segmentation and recognition of symbols within handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 4, pages 2439–2442. IEEE, 1995. 17, 34, 80

A. Kosmala and G. Rigoll. On-line handwritten formula recognition using statistical methods. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 2, pages 1306–1308. IEEE, 1998. 80

R. Kremer. Visual languages for knowledge representation. In *Proc. of 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98) Banff, Alberta, Canada. Morgan Kaufmann*, 1998. 15

K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43, 1990. 48

S. Lehmberg, H.-J. Winkler, and M. Lang. A soft-decision approach for symbol segmentation within handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 6, pages 3434–3437. IEEE, 1996. 34

T. Lin, B. G. Horne, P. Tino, and C. L. Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996. 48

R. Maalej and M. Kherallah. Improving mdlstm for offline arabic handwriting recognition using dropout at different positions. In *International Conference on Artificial Neural Networks*, pages 431–438. Springer, 2016. 95

R. Maalej, N. Tagougui, and M. Kherallah. Recognition of handwritten arabic words with dropout applied in mdlstm. In *International Conference Image Analysis and Recognition*, pages 746–752. Springer, 2016. 95

S. MacLean and G. Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *International Journal on Document Analysis and Recognition (IJDAR)*, 16(2): 139–163, 2013. 9, 19, 34, 37, 38

K. Marriott, B. Meyer, and K. B. Wittenburg. A survey of visual language specification and recognition. In *Visual language theory*, pages 5–85. Springer, 1998. 15

W. A. Martin. Computer input/output of mathematical expressions. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 78–89. ACM, 1971. 16, 34

N. E. Matsakis. *Recognition of handwritten mathematical expressions*. PhD thesis, Massachusetts Institute of Technology, 1999. 10, 17, 34, 80

R. Messina and J. Louradour. Segmentation-free handwritten chinese text recognition with lstm-rnn. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 171–175. IEEE, 2015. 95

H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain. Icfhr 2014 competition on recognition of on-line handwritten mathematical expressions (crohme 2014). In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 791–796. IEEE, 2014. 71, 89, 111, 117

H. Mouchère, R. Zanibbi, U. Garain, and C. Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the crohme competitions, 2011–2014. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(2):173–189, 2016. 16, 30, 34

M. C. Mozer. Induction of multiscale temporal structure. In *Advances in neural information processing systems*, pages 275–282, 1992. 48

F. Á. Muñoz. *Mathematical Expression Recognition based on Probabilistic Grammars*. PhD thesis, 2015. 80, 83, 100

D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011. 33

A. Robinson and F. Fallside. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987. 47

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. 44, 46

J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992. 48

M. Schuster. On supervised learning from sequential data with applications for speech recognition. *Daktaro disertacija, Nara Institute of Science and Technology*, 1999. 47

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997. 47

S. Smithies, K. Novins, and J. Arvo. A handwriting-based equation editor. In *Graphics Interface*, volume 99, pages 84–91, 1999. 80

K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015. 10, 11, 51, 52, 95, 96

E. Tapia. *Understanding mathematics: A system for the recognition of on-line handwritten mathematical expressions*. PhD thesis, Freie Universität Berlin, 2005. 17, 34

E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In *International Workshop on Graphics Recognition*, pages 329–340. Springer, 2003. 17, 34

E. Tapia and R. Rojas. A survey on recognition of on-line handwritten mathematical notation. 2007. 16, 34

K. Toyozumi, N. Yamada, T. Kitasaka, K. Mori, Y. Suenaga, K. Mase, and T. Takahashi. A study of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 630–633. IEEE, 2004. 34

S. H. Unger. A global parser for context-free phrase structure grammars. *Communications of the ACM*, 11 (4):240–247, 1968. 37, 39

P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988. 44, 47

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990. 47

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486, 1995. 46, 47

H.-J. Winkler and M. Lang. Online symbol segmentation and recognition in handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 4, pages 3377–3380. IEEE, 1997a. 80

H.-j. Winkler and M. Lang. Symbol segmentation and recognition for understanding handwritten mathematical expressions. In *Progress in handwriting recognition*. Citeseer, 1997b. 80

H.-J. Winkler, H. Fahrner, and M. Lang. A soft-decision approach for structural analysis of handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 4, pages 2459–2462. IEEE, 1995. 17, 34

R. Yamamoto, S. Sako, T. Nishimoto, and S. Sagayama. On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. In *tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006. 9, 19, 34, 35

S. Yu, L. HaiYang, and F. K. Soong. A unified framework for symbol segmentation and recognition of handwritten mathematical expressions. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 854–858. IEEE, 2007. 34, 80

R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012. 9, 16, 27, 29, 34, 125

R. Zanibbi, D. Blostein, and J. R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on pattern analysis and machine intelligence*, 24(11):1455–1467, 2002. 17, 34

R. Zanibbi, H. Mouchère, and C. Viard-Gaudin. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In *IS&T/SPIE Electronic Imaging*, pages 865817–865817. International Society for Optics and Photonics, 2013. 27, 31, 33

J. Zhang, J. Du, S. Zhang, D. Liu, Y. Hu, J. Hu, S. Wei, and L. Dai. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. *Pattern Recognition*, 2017. 34, 39

L. Zhang, D. Blostein, and R. Zanibbi. Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 972–976. IEEE, 2005. 17, 34

X. Zhu, P. Sobhani, and H. Guo. Dag-structured long short-term memory for semantic compositionality. In *Proceedings of NAACL-HLT*, pages 917–926, 2016. 51, 96

X.-D. Zhu, P. Sobhani, and H. Guo. Long short-term memory over recursive structures. In *ICML*, pages 1604–1612, 2015. 51, 95, 96

Publications of author

Journals

1. Zhang T, Mouchère H, Viard-Gaudin C. *A Tree-BLSTM based Recogniton System for Online Handwritten Mathematical Expression.* Submitted to International Journal on Document Analysis and Recognition (IJDAR), 2017.
2. Zhang T, Mouchère H, Viard-Gaudin C. *Using BLSTM for interpretation of 2-D languages.* Document numérique, 2016, 19(2): 135-157.

Conferences

1. Zhang T, Mouchère H, Viard-Gaudin C. *Tree-based BLSTM for mathematical expression recognition.* Document Analysis and Recognition (ICDAR), 2017 International Conference on, accepted.
2. Zhang T, Mouchère H, Viard-Gaudin C. *Online Handwritten Mathematical Expressions Recognition by Merging Multiple 1D Interpretations.* Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on. IEEE, 2016: 187-192.
3. Zhang T, Mouchère H, Viard-Gaudin C. *Using BLSTM for Interpretation of 2D Languages - Case of Handwritten Mathematical Expressions.* l'Ecrit et le Document (CIFED), 2016 Colloque International Francophone sur. 2016 CORIA-CIFED : 217-232.

Workshops

1. Zhang T, Mouchère H, Viard-Gaudin C. *On-line Handwritten Isolated Symbol Recognition using Bidirectional Long Short-term Memory (BLSTM) Networks.* Information and Communication Technologies, 2015 3th Sino-French Workshop on.

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Ting ZHANG

Nouvelles architectures pour la reconnaissance des expressions mathématiques manuscrites

New Architectures for Handwritten Mathematical Expressions Recognition

Résumé
Véritable challenge scientifique, la reconnaissance d'expressions mathématiques manuscrites est un champ très attractif de la reconnaissance des formes débouchant sur des applications pratiques innovantes. En effet, le grand nombre de symboles (plus de 100) utilisés ainsi que la structure en 2 dimensions des expressions augmentent la difficulté de leur reconnaissance. Dans cette thèse, nous nous intéressons à la reconnaissance des expressions mathématiques manuscrites en-ligne en utilisant de façon innovante les réseaux de neurones récurrents profonds BLSTM avec CTC pour construire un système d'analyse basé sur la construction de graphes. Nous avons donc étendu la structure linéaire des BLSTM à des structures d'arbres (Tree-Based BLSTM) permettant de couvrir les 2 dimensions du langage. Nous avons aussi proposé d'ajouter des contraintes de localisation dans la couche CTC pour adapter les décisions du réseau à l'échelle des traits de l'écriture, permettant une modélisation et une évaluation robustes. Le système proposé construit un graphe à partir des traits du tracé à reconnaître et de leurs relations spatiales. Plusieurs arbres sont dérivés de ce graphe puis étiquetés par notre Tree-Based BLSTM. Les arbres obtenus sont ensuite fusionnés pour construire un SLG (graphe étiqueté de traits) modélisant une expression 2D. Une différence majeure par rapport aux systèmes traditionnels est l'absence des étapes explicites de segmentation et reconnaissance des symboles isolés puis d'analyse de leurs relations spatiales, notre approche produit directement un graphe SLG. Notre système sans grammaire obtient des résultats comparables aux systèmes spécialisés de l'état de l'art.

Mots clés
Reconnaissance d'expressions mathématiques, Réseaux de neurones récurrents, BLSTM, Écriture en ligne.

Abstract
As an appealing topic in pattern recognition, handwritten mathematical expression recognition exhibits a big research challenge and underpins many practical applications. Both a large set of symbols (more than 100) and 2-D structures increase the difficulty of this recognition problem. In this thesis, we focus on online handwritten mathematical expression recognition using BLSTM and CTC topology, and finally build a graph-driven recognition system, bypassing the high time complexity and manual work in the classical grammar-driven systems. To allow the 2-D structured language to be handled by the sequence classifier, we extend the chain-structured BLSTM to an original Tree-based BLSTM, which could label a tree structured data. The CTC layer is adapted with local constraints, to align the outputs and at the same time benefit from introducing the additional 'blank' class. The proposed system addresses the recognition task as a graph building problem. The input expression is a sequence of strokes, and then an intermediate graph is derived considering temporal and spatial relations among strokes. Next, several trees are derived from the graph and labeled with Tree-based BLSTM. The last step is to merge these labeled trees to build an admissible stroke label graph (SLG) modeling 2-D formulas uniquely. 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 any grammar, achieves competitive results in online math expression recognition domain.

Key Words
Mathematical expression recognition, recurrent neural networks, BLSTM, online handwriting.