6-2014

Intelligent Combination of Structural Analysis Algorithms: Application to Mathematical Expression Recognition

Amit Pillay

Intelligent Combination of Structural Analysis Algorithms: Application to Mathematical Expression Recognition

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Richard Zanibbi, Supervisor

Roxanne Canosa, Reader

Zack Butler, Observer

Intelligent Combination of Structural Analysis Algorithms: Application to Mathematical Expression Recognition

by

Amit Pillay, B.E.

THESIS

Presented to the Faculty of the Department of Computer Science

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

Rochester Institute of Technology

June 2014

Acknowledgments

I am greatful to many people who have helped me directly or indirectly in completing my thesis. I am thankful to the department of Computer Science of Rochester Institute of Technology for giving me the opportunity to study my Masters. I am thankful for financial support from Xerox corporation through CAT-EIS/NYSTAR grant.

I am highly grateful to Dr. Richard Zanibbi for his excellent supervision over the past few years. He has been a great mentor and inspiration. It has been absolute pleasure working under him and imbibing his wisdom. I would also like to thank Dr. Anurag Agarwal for his assistance in clearing certain mathematical problems.

Last but certainly not the least, I would like to thank my parents- Arun Pillay and Uma Pillay, for their unconditional love and support throughout my life without whom I would not be here to acheive whatever I have acheived, my brother Rohit Pillay with whom I have shared the most fun part of my life and my loving wife Shruti Lakkaraju who has been a pillar of strength in my life this past year.

Abstract

Intelligent Combination of Structural Analysis Algorithms: Application to Mathematical Expression Recognition

Amit Pillay, M.S.
Rochester Institute of Technology, 2014

Supervisor: Dr. Richard Zanibbi

Structural analysis is an important step in many document based recognition problem. Structural analysis is performed to associate elements in a document and assign meaning to their association. Handwritten mathematical expression recognition is one such problem which has been studied and researched for long. Many techniques have been researched to build a system that produce high performance mathematical expression recognition. We have presented a novel method to combine multiple structural recognition algorithms in which the combined result shows better performance than each individual recognition algorithms. In our experiment we have applied our method to combine multiple mathematical expression recognition parsers called DRACULAE. We have used Graph Transformation Network (GTN) which is a network of function based systems in which each system takes graphs as input, apply function and produces a graph as output. GTN is used to combine multiple DRACULAE parsers and its parameter are tuned using gradient based learning.

It has been shown that such a combination method can be used to accentuate the strength of individual algorithms in combination to produce better combination result which higher recognition performance. In our experiment we were able to obtain a highest recognition rate of 74% as compared to best recognition result of 70% from individual DRACULAE parsers. Our experiment also resulted into a maximum of 20% reduction of parent recognition errors and maximum 37% reduction in relation recognition errors between symbols in expressions.

Table of Contents

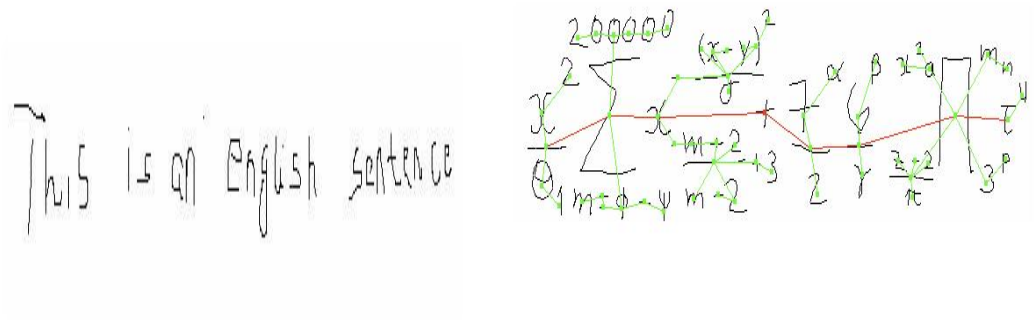
List of Tables

List of Figures

Chapter 1

Introduction

Automated mathematical expression recognition is of great practical importance. If technology matures printed document that needs to be converted to electronic form can be done automatically. Recognition of handwritten expressions can be useful with data tablet which can be an alternative and convenient way of writing rather than writing a LATEX expression [28].



(a) (b)

Figure 1.1: Symbol layout in a textline vs. within a mathematical expression (from Tapia and Rojas [25]). At right the dominant baseline of the handwritten expression is shown in red.

Automatic recognition of handwritten or typeset mathematical expression has been an important area of research for over many years [4][8]. Generally, mathematics recognition

involves two major steps: recognition of individual symbols present in the expression and interpreting the layout of the recognized symbols also known as structural analysis [4]. Structural analysis step may not be so crucial for recognition problems that involve only standard english character 1.1(a). However for problems like recognition of mathematical expressions, the actual position and location of symbols is important and their semantics depends on their arrangement as shown in Figure 1.1(b).

Although there has been number of research that focus on applying different methods of structural analysis [15] [19] [13] [10], every system has improvement limitation in recognition. It has been understood that for most tasks we are faced with a ceiling to the improvement that can be reached with any available machine learning system [27]. [27] explained that the potential loophole of any recognition algorithm is that each of them introduce their own inductive bias to the recognition task and produce different errors. An earlier study on system combination was done by Ali and Pazzani [1] in 1996 who studied the effect of using combining different evidence descriptions of a class on the classification error rate. The result show significant reduction in error rate for half of the different data set used for the experiment and for most of the other set error remains the same.

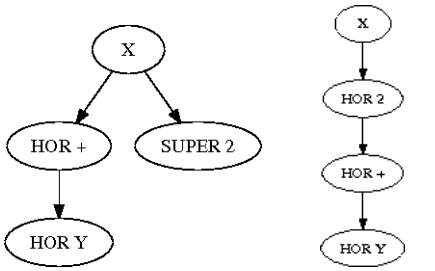
Hence combination techniques can be adopted to overcome the individual system limitations to further improve the recognition rate. Over the decade, there have been number of research on combination techniques in Natural Language Processing [17] [27] [22] [12], in speech recognition [11], in lexical disambiguation [6] and named entity recognition [5]. We proposed a novel combination technique to combine structural recognition algorithms for parsing two dimensional notations or languages like mathematical expression.

Thesis Statement: Recognition performance of structural analysis parsers like mathematical recognition parsers can be improved by adopting intelligent parser combination method.

Our combination technique uses Graph Transformer Network(GTN) which was previously used in handwritten character recognition described by LeCun et. al [3]. [3] defined a generalized way in which a series of modules representing key tasks like feature extraction or segmentation in recognition process can be trained. Each module in a *Graph Transformer Network* (GTN) takes as input directed graphs and outputs a directed graph, where the graphs have numerical attributes used for learning, and the production of the output is learnt using a gradient descent algorithm. The power of the GTN is that it can combine heterogeneous and dynamic modules and train them using a global optimization criterion, rather than rely upon localized optimization [3].

In GTNs, the produced graphs contain nodes and/or edges that carry numerical attributes, which are used for learning parameters in the GTN (for example, penalties for a relationship defined by edges). During the learning process, labels are used to assign penalties against edges based upon a loss function. These penalties are then backpropagated to adjust weightings in the previous layer of GTN modules. We can see that for labeled graphs that there will be some overlap in their structure to other unlabeled graphs. For example, if our graph is describing the layout of symbols in a mathematical expression such as $X^2 + Y$ (see Figure 1.2), then we can see that the graph will be similar to that for $X + Y$ or X^{2+Y}. Using this similarity, we can therefore partially label these additional graphs and calculate appropriate penalties for the edges. This thesis presents such an algorithm which

$X^2 + Y$



(a) Tree 1 (b) Tree 2

Figure 1.2: Two Baseline Structure Trees representing Interpretations of $X^2 + Y$

uses information regarding the similarity of graphs, and an appropriate (and differentiable) penalty function to assign penalties for the labeled and unlabeled edges.

Chapter 2

Background

Mathematical Expressions MEs form an essential part of scientific and technical documents. Mathematical Expressions can be typeset or handwritten which uses two dimensional arrangements of symbols to transmit information. Recognizing both forms of mathematical expressions are challenging.

Generally speaking understanding and recognizing mathematical expression, whether typeset or handwritten, involves three activities: Expression localization, symbol recognition and symbol-arrangement analysis. ME localization involves finding and extracting mathematical expression from the document. Symbol recognition converts the extracted expression image into a set of symbols and symbol arrangement analyzes the spatial arrangement of set of symbols to recover the information content of the given mathematical notations.

Now based on the recognition process, symbol recognition activity can further subdivided as 1) preprocessing - noise reduction, deskewing, slant correction etc, 2) segmentation to isolate symbols 3) and finally, recognition. Similarly depending upon the symbol-arrangement algorithm, symbol arrangement analysis can be further subdivided into a) identification of spatial relationships among symbols b) identification of logical relationships among symbols 3) construction of meaning. These processes can be executed in series or in

parallel with latter processes providing contextual feedback for the earlier processes. The order of these recognition activities can vary somewhat, for example, partial identification of spatial and logical relationships can be performed prior to symbol recognition.

2.1 Preprocessing

Preprocessing is required to eliminate irregularities and noise from the image, especially in handwritten character recognition. Certain preprocessing method requirements may depend upon the techniques used for recognition. [16] uses chain code method for handwritten image representation. Preprocessing involves slant angle correction in which global slant angle from different vertical lines is estimated and tangent of the estimated global slant angle is used to correct for slant. Smoothing of image involves elimination of small blobs (noise) on the contour. A sliding 3-component one dimensional window is applied overall components during which components are removed or added based on the orientation of components. Average stroke width is estimated by dividing chain code contours horizontally and by tracing left to right various distances between outer and inner contour. [18] performs size normalization to reduce variation in character size. To avoid significant deformation due to directly scaling of all images to identical size, a holistic approach is used for scaling in which if width/height ratio is less than 0.8 then scale is identical horizontally and vertically otherwise the scale factor is set to 0.8 to prevent large variation in image width.

2.2 Character segmentation

Character segmentation, next step in ME recognition, has long been a critical area of OCR process. Depending upon the requirement, character segmentation techniques is divided into four major headings [21]. Classical approach of segmentation also called dissection technique consists of partitioning the input image into sub-images based on their inherent features, which are then classified. Another approach to segmentation is a group of techniques that avoids dissection and segments to image either explicitly by classification of pre-specified windows, or implicitly by classification of subsets of spatial features collected from the image as a whole. Another approach is a hybrid approach employing dissection but using classification to select from admissible segmentation possibility. Finally holistic approach avoids segmentation process itself and performs recognition entire character strings.

Various techniques have been used for segmentation that involves dissection. White spaces between the characters are used to detect segmentation points. Pitch which is the number of characters per unit of horizontal distance provides a basis for estimating segmentation points. The segmentation points obtained for a given line should be approximately equally spaced at the distance that corresponds to pitch [21].

Inter-character boundaries can be obtained if most segmentation takes place by finding columns of white. Now all segmentation points that do not lie near these boundaries can be rejected as caused due to broken characters. Similarly we can estimate missed points due to merged characters. Hoffman and McCullough gave a framework for segmentation that involves three steps i.e. 1) Detection of the start of the character, 2) A decision to begin testing for the end of a character called sectioning, 3) Detection of end-of-character.

Sectioning is done by weighted analysis of horizontal black runs completed versus run still incomplete. Once sectioning determines the regions of segmentation, rules were invoked to segment based on either an increase in bit density or the use of special features designed to detect end-of-character.

In [2], segmentation in cursive handwritten characters is performed in the binary word image by using the contour of the writing. Determination of segmentation regions is done in three steps. In first step a straight line is drawn in the slant angle direction from each local maximum until the top of the word image. While going upward in the slant direction, if any contour pixel is hit, this contour is followed until the slope of the contour changes to the opposite direction. An abrupt change in the slope of the contour indicates an end point. A line is drawn from the maximum to the end point and path continues to go upward in slant direction until the top of the word image. In step 2, a path in the slant direction from each maximum to the lower baseline, is drawn. Step 3 follows the same process as in step 1 in order to determine the path from lower baseline to the bottom of the word image. Combining all the three steps gives the segmentation regions. In [16] segmentation involves detecting ligatures as segmentation points in cursive scripts. Alternatively, concavity features in the upper contour and convexities in the lower contour are used in conjunction with ligatures to reduce the number of potentials segmentation points.

Another dissection technique that applies to non-cursive characters is bounding box technique [21]. In this analysis, the adjacency relationships between characters are tested to perform merging or their size or aspect ratios are calculated to trigger splitting mechanisms. Another involves splitting of connected components. Connected components are merged or

split according to rules based on height and width of the bounding boxes. Intersection of two characters can give rise to special image features and different dissection methods have been developed to detect these features and to use them in splitting a character string images into sub-images.

[9] focuses on segmentation of single and multiple touching character segmentation. [9] proposes a new technique that links the feature points on the foreground and background alternately to get the possible segmentation path. Gaussian mixture probability function is determined and used to rank all the possible segmentation paths. Segmentation paths construction is performed separately for single touching characters and for multiple touching characters. All the paths from to two analysis are collectively processed to remove useless strokes and then mixture Gaussian probability function is applied to decide which on is the best segmentation path.

Another kind of approach to character segmentation is recognition based approach. In these segmentation processes letter segmentation is a by-product of letter recognition. The basic principle is use a mobile window of variable width to provide sequences of tentative segmentation which are confirmed (or not) by character recognition. A technique called Shortest Path Segmentation selects the optimal combination of cuts from the predefined set of candidate cuts that construct all possible legal segments through combination. A graph whose nodes represent acceptable segments is the created. The paths of these graphs represent all legal segmentations of the word. Each node of the graph is then assigned a distance obtained by the neural net recognizer. The shortest path though the graph thus corresponds to the best recognition and segmentation of the word. An alternative method attempts to match subgraphs of features with predefined character prototypes.

Different alternative are represented by a directed network whose nodes correspond to the matched subgraphs. Word recognition is done by searching for the path that gives the best interpretation of the word features [9].

2.3 Symbol-Arrangement Analysis

One approach to symbol-arrangement analysis is syntactic approach. Syntactic approach makes use of two dimensional grammar rules to define the correct grouping of math symbols. Co-ordinate grammar for recognition is presented by Anderson. The grammar specifies syntactic rules that subdivide the set of symbols into several subsets, each with its own syntactic subgoal. The final interpretation result is given by the m attribute of the grammar start's symbol where m represents ASCII encoding of the meaning of symbol-set. Although coordinate grammar provides a clear and well structured recognition approach, its slow parsing speed and difficulty to handle errors are its major drawbacks. In [8], a syntactic approach is adopted in which a system consisting of hierarchy of parsers for the interpretation of 2-D mathematical formulas is described. The ME interpreter consists of two syntactic parser top-down and bottom-up. It starts with a priority operator in the expression to be analyzed and tries to divide it into sub-expressions or operands which are then analyzed in the same way and so on. The bottom-up parser chooses from the starting character and from the neighboring sub-expressions the corresponding rule in the grammar. This rule gives instructions to the top-down parser to delimit the zones of neighboring operands and operators.

Garain and Chaudhari in [14], proposes a two pass approach to determine arrangement of symbols. The first pass is a scanning or lexicon analysis that performs micro-level

examination of the symbols to determine the symbol groups and to determine their categories or descriptors. The second pass is parsing or syntax analysis that processes the descriptors synthesized in the first pass to determine the syntactical structure of the expression. A set of predefined rules guides the activities in both the passes.

Another symbol-arrangement analysis approach is recursive projection profile cutting[20]. Cutting by the vertical projection profile is attempted first, followed by horizontal cuts for each resulting regions. The process repeats until no further cutting is possible. The resulting spatial relationships are represented by a tree structure. Although the method looks simple and efficient technique, it is still under study and also involves additional processing for symbols like square root, subscripts and superscripts as these can be handled by projection profile cut.

Another approach discussed is the Graph Rewriting. Graph rewriting involves information represented as an attributed graph and the graph get updated through the application of graph-rewriting rules. An initial graph contains one node to represent each symbol, with nodes attributes recording the spatial coordinates of the symbol. Graph rewriting rules are applied to add edges representing meaningful spatial relationships. Rules are further applied to prune or modify these edges identifying logical relationships from the spatial relationships. In [15], Ann Grbavec and Dorothea Blostein proposed a novel-graph rewriting techniques that addresses the recursive structure of mathematical notations, the critical dependence of the meaning upon operator precedence and the presence of ambiguities that depends upon global context. The recognition system proposed called EXPRESSO, is based on Build-Constrain-Rank-Incorporate model where the Build phase constructs edges to represent potentially meaningful spatial relation- ships between symbols. The Constrain phase

applies information about the notational conventions of mathematics to remove contradictions and resolve ambiguities. The Rank phase uses information about the operator precedence to group symbols into sub-expressions and the Incorporate phase interprets sub-expressions.

Twaakyondo and Okamoto [26] discuss two basic strategies to decide the layout of structure of the given expression. One strategy is to check the local structures of the sub-expressions using a bottom-up method (specific structure processing). It is used to analyze nested structures like subscripts, superscripts and root expressions. The other strategy is to check the global structure of the whole expression by a top-down method (fundamental structure processing). It is used to analyze the horizontal and vertical relations between sub-expressions. The structure of the expression is represented as a tree structure.

Chou in [10] proposed a two-dimensional stochastic context-free grammar for recognition of printed mathematical expressions. The recognized symbols are parsed with the grammar in which each production rule has an associated probability. The main task of the process is to find the most probable parse tree for the input expression. The overall probability of a parse tree is computed by multiplying together the probabilities for all the production rules used in a successful parse.

2.4 Baseline Structure Tree

Layout of symbols in mathematical expressions are organized into number of baselines. Layout analysis of mathematical expression extract these baselines, which can be arranged in a structure called Baseline Structure Tree (BST) [28]. The recognized symbols with bounding box coordinates form the input to our GTN structure. Layout parsers in

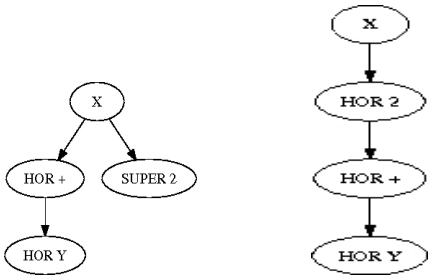
GTN generate BST based on these input. Each node in BST represent the symbol in mathematical expression and edges represents spatial association between the symbols. Parser also associate a penalty with each of the spatial association.

2.5 DRACULAE Parser

For mathematical expression parsing we made use of DRACULAE parser [28]. DRACULAE stands for Diagram Recognition Application for Computer Understanding of Large Algebraic Expressions. DRACULAE analyzes symbol layout in mathematical expressions by searching for baselines in the input expressions. These baselines are arranged into BSTs. These BSTs are processed and restructured in three passes of recognition process adopted in DRACULAE. Layout pass takes the lists of recognized symbols with their bounding boxes and arrange them to form a BST. Arrangement is done by identifyng dominant baseline in an expression and partitioning symbols not part of dominant baseline in separate regions relative to dominant baselines. Regions are identified as ABOVE, BELOW, SUPER, SUBSC, UPPER, LOWER, TLEFT, BLEFT, CONTAINS and EXPRESSIONS. The process of identifying dominant baseline and partitioning symbols accordingly is recursively performed among regions. Lexical analysis pass identifies compound symbols in BST and groups them together using a set of rules. Lexical analysis is done based on local adjacency of symbols combining those symbols that are adjacent to each other and matches one of the known function name or pattern. Expression Analysis pass then applies mathematical expression grammar to convert Lexed BST into operator tree. Expression Syntax analysis uses the precedence and associativity of mathematical operators to create an expression parse tree. Expression Semantic analysis applies the symantic rules to identify implied operators

and ordering of operands according to operators.

2.6 Tree Edit Distance



(a) Tree1 (b) Tree2

Figure 2.1: Two Baseline Structure Trees Representing Interpretations of $X^2 + Y$

Tree Edit Distance(TED) is used as error metric to compute the error between a graph produced by layout analysis and the desired graph(Ground Truth). Suppose there a cost function defined on every operation (insertion, deletion or relabeling) that can be performed on trees, the tree edit distance is the minimum of sum of the cost of all the operations performed in process of turning a tree T1 to Tree T2. Tai in [24] gave an algorithm for computing tree edit distance between two labeled ordered trees that is similar to a string to string correction algorithm. Preorder traversal of trees are performed to label each node uniquely in a tree. Tai then perform mapping between nodes of the two trees to establish which operations are performed to transform from tree T1 to tree T2. Edit distance between two trees is then determined by the minimum cost mapping. The algorithm solves the problem in time O(V1*V2*$L1^2 * L2^2$), where V1 and V2 are the numbers of nodes respectively of T1 and T2, and L1 and L2 are the maximum depths respectively of T1 and T2.

Our TED algorithm involves computing the tree edit distance between two Baseline Structure Trees (BST) [28] as shown in Figure 2.1(a) and Figure 2.1(b) representing two interpretations of a mathematical expression having matching symbols but different spatial relationships between them. The expressions are converted into directed trees having nodes with values as a symbol in given expression concatenated with the type of spatial relationship which the symbol has with its parent symbol as shown in Figure 2.1(a) and Figure 2.1(b). Tree edit distance is then given as the number of spatial relationship mismatches between nodes of two trees plus the number of mismatching parents of nodes in two trees. In the given example, Tree2 shows the incorrect interpretation of the expression $X^2 + Y$. The tree edit distance between Figures 2.1(a) and 2.1(b) is 2. One is due to symbol "2" in Tree2 having a mismatched spatial association with its parent "X" (i.e HOR instead of SUPER as in Tree1) and one due to symbol "+" having mismatching parent (have parent as "2" instead of "X" as in Tree1). Edit distance is then normalized as number of edits / 2n(where n is the number of symbols); 2n is the maximum edit distance, where every symbol has the wrong parent, and in the wrong spatial relationship. Normalization of edit distance is essential in order to meaningfully rank the edits(error) with respect to number of symbols in the expressions that are been compared. To cope with the repeated symbols in the expression symbols needs to be numbered to identify them uniquely.

We presented a paper [29] at ICDAR 2011 conference that introduces a stroke based performance metric to evaluate the performance of handwritten mathematical expressions. The paper presents a metric that takes the hollistic approach from strokes to spatial structure recognition while evaluating the performance. In this technique, math expressions are represented through bipartite graphs with nodes representing individual strokes in the

expression(described in the paper). The edges in the graph represents the relationships between a node in the graph and its ancestor nodes in the symbol layout tree. Given such bipartite graph representations of recognizer output and ground truth, we compute the recognition error by the number of disagreements in terms of stroke labels and spatial relationships which represents distance measure between the two representations.

As there is only one metric involved in evaluating a recognition algorithm from segmentation, classification to structural layout, such a metric can be used to better compare mathematical recognition algorithms as well as provide a single learning function to improve a complete recognition system.

2.7 Parser combination

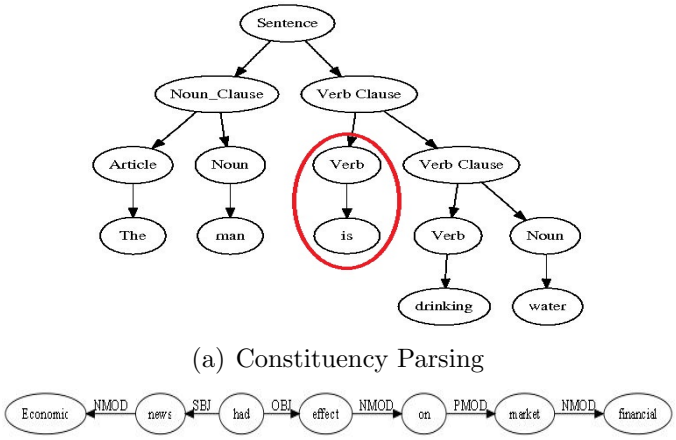
Classification problem can be solved by having a descriptor for each class which is based on different features taken under consideration and then classifying data to a class based on how close the features of data are to the descriptor of that class. These class-defining descriptors can be formed from a single model or can be formed by combination of mutiple models together called as ensemble of models.

Ali and Pazzani [1] performed combination of multiple models in an ensemble for classification. One of the hypotheses of their experiment was to see the effect of multiple descriptors of a class on classification error in comparison with error from single descriptor. They conducted the experiment on number of domains like Tic-Tac-Toe, DNA and wine. The experiment invovled different combination techniques, such as Uniform Voting that consist of counting of each class number of descriptors have got atleast one rule satisfied, Bayesian Combination that involves assignment of class having highest posterior probability,

Distribution Summation that involves summation of vectors of all distribution of all satisfied rules from all models and so on [1]. The results shows a statistically significant reduction in error rate for almost all of the data set used or no increase in some cases. For some dataset like wine, the recognition accuracy has increased by factor of 6 from 93.3 percent (with single descriptor) to 98.9 percent (with combined descriptors). This statistically significant reduction in error rate has proven to be true for all the combination techinique tried for the experiment[1].

Parser combination involves combining parse trees produced by different parsers, with the goal of reducing error. Parser combination has been explored in a number of areas. Syntactic parsing of a sentence in Natural Language Processing(NLP) can be done in two different ways i.e Constituent Parsing and Dependency Parsing (Figure 2.2). Constituent parsing involves breaking down of sentence into number of syntactic constituents (Noun Clause, Verb Clause or Preposition Clause). Dependency parsing, on the other hand, identify dependencies between words in a sentence and connect the words with that dependency. Our mathematical expression parsing technique is a dependency parsing rather than constituency parsing as the technique involves identifying dependencies between symbols in an expression.

Combination of parser can be performed in two different ways as mentioned by Henderson and Brill [17]. The first method is *Parse Hybridization* in which constituents from each parser's output are recombined to construct a improved parse. The second is *Parser Switching* in which a switching technique is applied which selects constituents using simple majority voting. For example, if a constituent is included in at least half of the parse trees to combine, the constituent becomes the part of combined tree. As shown in



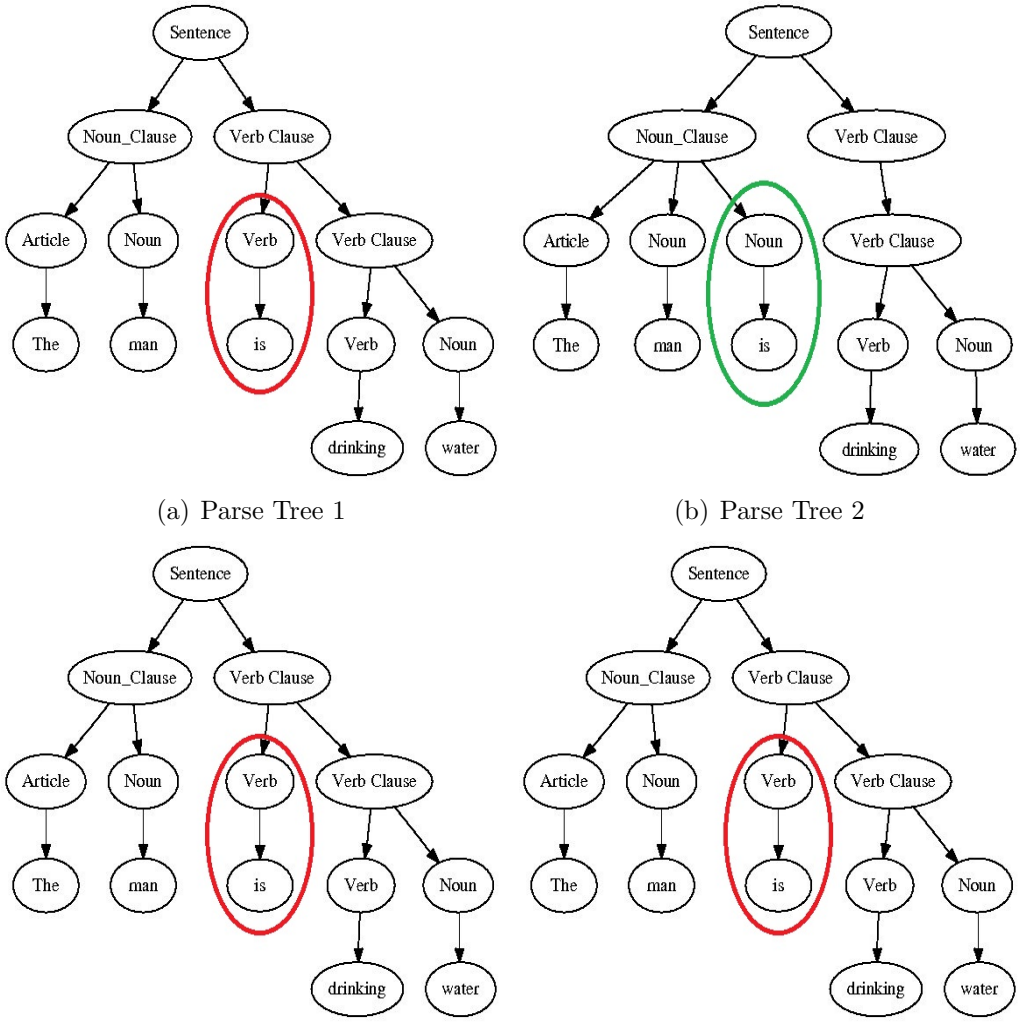
(a) Constituency Parsing

(b) Dependency Parsing

Figure 2.2: Types of Syntactic Parsing

the following Figure 2.3, the constituent *Verb* is included in two of three parse trees to be combined (tree 1 and tree 3), and so it becomes the part of combined tree. Their experiment showed that these combination techniques gave better recognition results than any of the individual parser used for the experiment. Also the combination technique degrade very less if a poor parser is added to the set.

[22] explained parser combination also based on parse hybridization. Parser combination for both dependency parsing and constituent parsing are explained. In dependency reparsing different dependency structures from different parsers are obtained and then a graph is contructed with each word as node and weighted directed edges as dependencies obtained from initial dependency structures. This weighted graph is parsed again finding maximum spanning tree to obtain well-formed optimal combined dependency structure. Whereas in constituent reparsing, each individual parses are broken down into contituents with their weights in a chart. Weights for identical constituents from parses are added



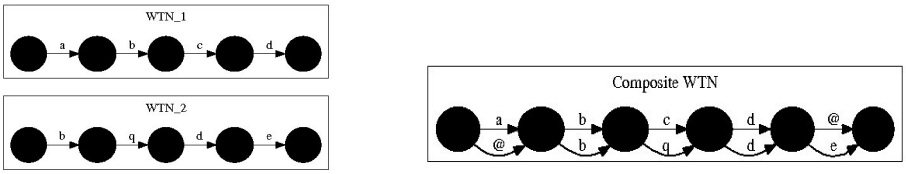
(a) Parse Tree 1 (b) Parse Tree 2

(c) Parse Tree 3 (d) *Combined Tree*

Figure 2.3: Combination (d) of parse trees for English text (a-c)

together and finally running a bottom-up parsing algorithm against this chart to obtain maximized well-formed combined parse tree.

Francis Brunet-Manquet in [7] discussed a technique for parser combination for NLP. The combination, however, is applied to dependency parsers unlike the constituent parsers used by Henderson and Brill. The method involves first extracting linguistic data along with confidence rates from a sentence extracted by different syntactic parsers. These linguistic data are then arranged in a respective single structure called dependency structure that identifies the dependencies between these data. From these structures, a common dependency structure is obtained consisting of linguistic data contained in different dependency structures along with different confidence rates associated by different parsers for each data. These confidence rate is nothing but a weighted vote of a parser for a data. Finally a grouping of all linguistic data is done by calculating a combined confidence rate for each data. Combined rate is given as the sum of confidence rates of a data divided by number of parsers that provided this data.



(a) WTNs (b) Composite WTNs

Figure 2.4: Combination of WTNs

Fiscus in [11] describes a combination technique that combines Automatic Speech Recognition (ASR) systems using two steps. Refer to Figures 2.4(a) and 2.4(b). In the first step, output from multiple ASRs which is in the form of Word Transition Networks

(WTN) are combined using dynamic programming alignment to produce a single WTN. The combination is done sequentially, combining first two and then adding each in turn to the current combined result. The next step is to extract a single WTN from the composite WTN using the voting or scoring scheme that selects the best scoring word sequence from the given hypotheses. One scheme uses frequency of occurrence of particular word type. Higher the frequency, better the score for the word. Other two schemes involve the use of confidence scores generated by ASRs for each word along with frequency scores in voting for best word sequence. One out of the two uses average confidence score while other uses the maximum confidence score for each word type. Experimental results showed that each scoring scheme results in improved word error(WE) reduction than individual ASR system with scheme involving maximum confidence score resulting in maximum WE reduction.

2.8 Graph Transformer Network

Our combination technique uses Graph Transformer Network (GTN) which was previously used in handwritten character recognition described by LeCun et. al. [3]. Graph Transformer Network (GTN) consists of network of modules that takes one or more graphs or trees as inputs and produce graphs as outputs. These modules are differentiable w.r.t its inputs and its internal parameters. Whole network is arranged just like feed-forward network of Artificial Neural Network(ANN) that minimizes a global error function. LeCun et. al showed that GTN can be trained using Gradient based back-propagation learning algorithm and it is just not limited to network like ANN that takes fixed sized vectors as input. Fixed sized vectors for data representation shows deficiency for tasks that deals with variable length inputs like continuous speech recognition and handwritten word recognition

[3]. Multiple length sequences which can be best represented in the form of graphs are quite often used for above mentioned tasks. Thus GTN learning can be used for these tasks representing data in the form of graphs in which each path represents a variable length sequence.

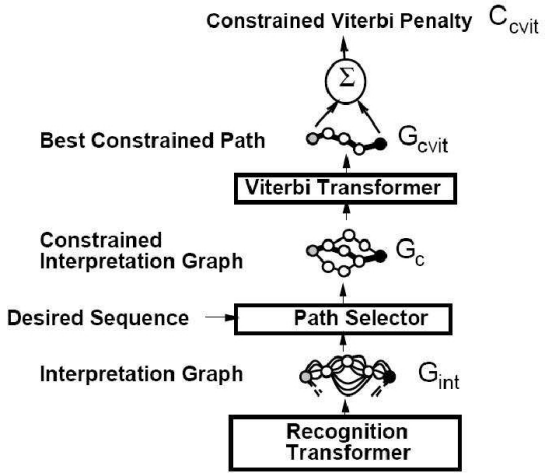


Figure 2.5: GTN for character recognition. [3]

Figure 2.5 shows a GTN for recognizing handwritten character string that uses gradient based learning.

2.8.0.1 GTN Training

GTN structure is trained as a whole with different modules interacting and propagating states and gradients. The given GTN takes a segmentation graph representing the segmented character image. Over-Segmentation of the image is performed so as to ensure that the resulting set of cuts contain the correct set of cuts of the given image. The given network makes use of two graph transformers called Recognition transformer and Vierbi transformer. Recognition transformer takes segmentation graph and produces Interpreta-

tion graph G_{int} in which every path represents a interpretation of a segmentation of the input [a word]. Each arc in G_{int} is associated with a class label and with a penalty for that class produced by the recognizer.

Another module called the Path Selector takes the G_{int} graph and the desired labelled sequence as input and selects paths from G_{int} that contains the desired label sequence to give Constrained graph G_c. Viterbi Transformer then applies Viterbi algorithm to the resulting graph to select a shortest path (a path with smallest cumulative penalty)C_{vit}. The loss function E to minimize is the average penalty of the correct lowest penalty path in the given graph. Gradient based learning is used to train the network. Gradient are computed at different differentiable modules of GTN which are propagated back to so as to compute their gradients and update module parameters.

Partial derivative of the error E for the best path with respect to edge penalties on the C_{vit} are equal to 1. This is because the loss function is simply the sum of the edge penalties on C_{vit}. Since the shortest path is subset of G_c, the partial derivatives of loss function E with respect to penalties on the arcs of G_c is 1 for those that appear in shortest path and 0 for those that do not. As path selector only selects those paths in G_{int} that have correct label sequence, arcs in G_{int} the appear in G_c have partial derivatives as 1 or 0, depending on whether that arc appear in shortest path G_{vit}. Now these penalties on G_{int} arcs are produced by individual output of each recognizer instance, therefore we have same gradients (1 or 0) for each output of each instance. Now these gradients present on each instance output is then propagated back to produce gradient w.r.t racognizer parameters. Let 'w' be the set of parameters of recognition transformer. For each recognizer instance of recognition transformer used to recognize a single character, a vector of partial derivatives of the loss

function E with respect to the recognizer instance parameter vector 'w' is obtained. Since recognizer instances are clones of each other, gradients of E w.r.t recognizer transformer parameter vector is computed as the sum of gradient vectors produced by each recognizer instance.

Chapter 3

Methodology

We describe a novel method of combining syntactic parsers. Our method is based on parse hybirdization techniques presented by Brill and Henderson in [17], in which constituents from each parser's output are recombined to construct a improved parse. Our method made use of GTN which consists of different instances of DRACULAE parsers [28] that produce different BSTs (parse trees). The edges on these parse trees are then penalized using penalty function which is described in section 3.1. Penalized edges then take part in combination process which is described in section 3.2. After combination, gradients with respect to combination error are computed using the process described in section 3.2.1. Different experiments are performed which is summerized under last section 3.2.2.

3.1 Relationship Penalty Function

Draculae identifies 7 different symbol classes into which different symbols can be classfied. They are as shown in the Table 3.1. Spatial regions around a symbol are partitioned based on the symbol class into seven regions: $SUPER$, $SUBSC$, $ABOVE$, $BELOW$, $UPPER$, $LOWER$ and $CONTAINS$.

Figure 3.2 shows different regions associated with each symbol class. A region is nothing but an axis-parallel box that includes only left and bottom edges [28]. Top and

Table 3.1: Class Membership [28]

Class	**Symbols**
Ascender	$0...9, A...Z$
Descender	$g, p, q, y, r, \eta, \rho$ $\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi$
Open Bracket	$[\{($
Non Scripted	Unary binary operators and relation $(\times, \setminus, \geq, \div, \equiv)$
Root	$\sqrt{\ }$
Variable Range	$\sum \prod \int \bigcap \bigcup$
Center	All other symbols

right edges are not included in region definition. Every symbol is reduced to a single point called the centroid of the symbol. A centroid of a symbol lies in exactly one region of its parent symbol.

DRACULAE determines spatial relationships between symbols based on the set of regions for a *parent* symbol, and the location of the centroid of the associated (child) symbol. Thus a child of a symbol whose symbol class is ascender can lie in either in $SUPER$ or $SUBSC$ or $ABOVE$ or $BELOW$ or HOR regions. For a pair of symbols that are associated in a ground truth baseline structure tree, our model defines a penalty based on how close the centroid of the neighboring symbol is to the vertical boundary (threshold) of these regions. For example, a symbol correctly assigned to the $SUPER$ region of another symbol will have a penalty based on how close that centroid is to the $SUPER$ region's threshold that separates the $SUPER$ and HOR regions.

Penalties are defined in the interval $[0, 1]$, using the sigmoid function:

$$P(y_\Delta) = \frac{1}{1 + e^{-by_\Delta}} \tag{3.1}$$

where P is the penalty e is constant, b is vertical scaling factor and y_Δ is the y-offset of the centroid of neighboring symbol vertical threshold. Vertical scaling factor is used to control

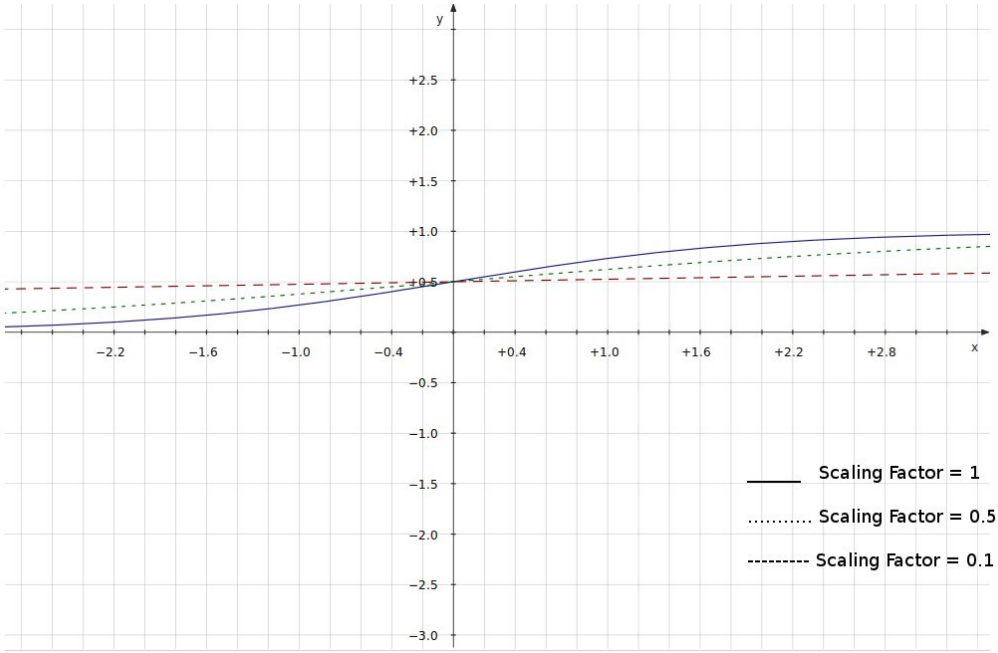


Figure 3.1: Sigmoid Function

the steepness of the sigmoid curve. When vertical scaling factor is high, the curve increases steeply which then act more as threshold and when vertical scaling factor is low (closes to zero), the curve flattens.

We describe penalty computation based on types of regions:

3.1.1 Upper and lower regions

Upper regions include $ABOVE$, $UPPER$ and $SUPER$ as shown in Figure 3.2.

For upper regions we defined y-offset as,

$$y_{\Delta} = (C - T_U)/(Y_{max} - Y_{min}) \tag{3.2}$$

where T_U is upper region's vertical threshold and C is the centroid on neighboring

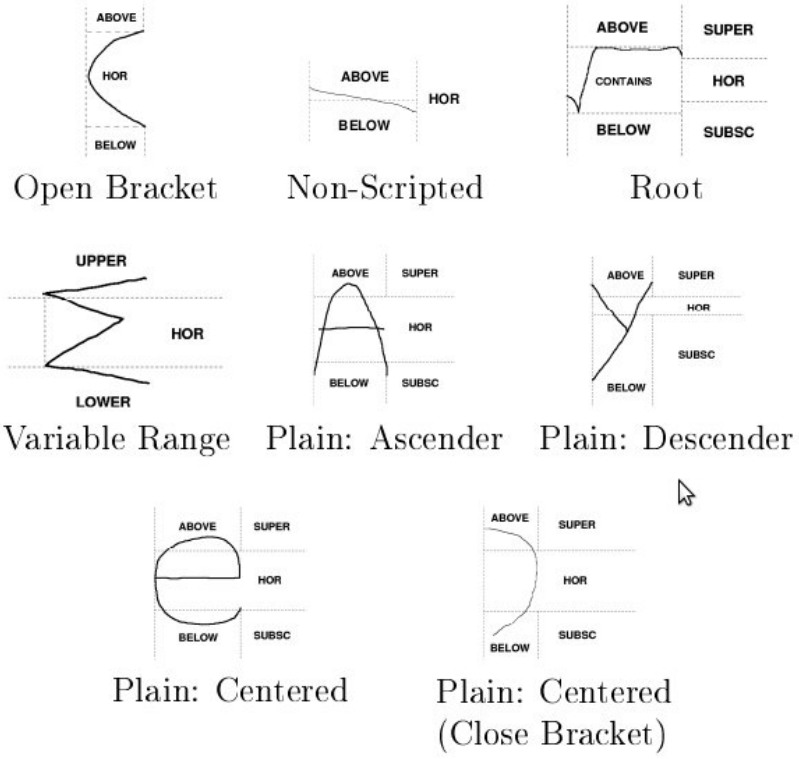


Figure 3.2: Regions [28]

symbol. We normalize the y-offset by dividing offset value by parent symbol's vertical size.

Similarly, for lower regions we defined y-offset as,

$$y_{\Delta} = (T_L - C)/(Y_{max} - Y_{min}) \tag{3.3}$$

where T_L is lower region's vertical threshold and C is the centroid on neighboring symbol.

So the penalty function for upper regions becomes:

$$P(y_{\Delta}) = \frac{1}{1 + e^{-b(C-T_U)/(Y_{max}-Y_{min})}} \tag{3.4}$$

Similarly the penalty function for lower regions becomes:

$$P(y_\Delta) = \frac{1}{1 + e^{-b(C-T_L)/(Y_{max}-Y_{min})}} \tag{3.5}$$

The upper penalty function (3.4) increases with the increase in y-offset. Whereas the lower penalty function have opposite effect, i.e. the lower penalty function (3.5) decreases with the increase in y-offset. This is described as below:

For upper penalty function:

$$P = \begin{cases} & < 0.5 \quad if\ y_\Delta < 0 \\ equation & 0.5 \quad if\ y_\Delta = 0 \\ & > 0.5 \quad if\ y_\Delta > 0 \end{cases}$$

For lower penalty function:

$$P = \begin{cases} > 0.5 \quad if\ y_\Delta < 0 \\ 0.5 \quad if\ y_\Delta = 0 \\ < 0.5 \quad if\ y_\Delta > 0 \end{cases}$$

3.1.2 Horizontal Regions

Horizontal regions include HOR and $CONTAINS$ as shown in Figure 3.2. For horizontal regions we first compute the centre of the distance between two vertical thresholds i.e upper threshold and its corresponding lower threshold. The y-offset computation is determined by location of neighboring symbols centroid. If the centroid lies above the centre, the offset from upper threshold is used to compute the penalty value and if the

centroid lies below the centre, the offset from lower threshold is uses to compute the penalty value. Let,

$$Centre = (T_U + T_L)/2 \tag{3.6}$$

if $C < Centre$, centroid lies towards upper vertical threshold, therefore we defined y-offset w.r.t upper threshold as,

$$y_\Delta = (T_U - C)/(Y_{max} - Y_{min}) \tag{3.7}$$

So the penalty function becomes:

$$P(y_\Delta) = \frac{1}{1 + e^{-b(T_U - C)/(Y_{max} - Y_{min})}} \tag{3.8}$$

Whereas. if $C > Centre$, centroid lies towards lower vertical threshold, therefore we defined y-offset w.r.t lower threshold as,

$$y_\Delta = (C - T_L)/(Y_{max} - Y_{min}) \tag{3.9}$$

So the penalty function becomes:

$$P(y_\Delta) = \frac{1}{1 + e^{-b(C - T_L)/(Y_{max} - Y_{min})}} \tag{3.10}$$

3.2 GTN Architecture

Input to our GTN is a list of mathematical symbols with bounding box co-ordinates. This symbol list is passed to one or more layout parsers. In this first experiment, we have used the publicly available DRACULAE parser [28] [1]. DRACULAE parsers have three

[1] in the Freehand Formula Entry System http://www.cs.rit.edu/ rlaz/ffes/

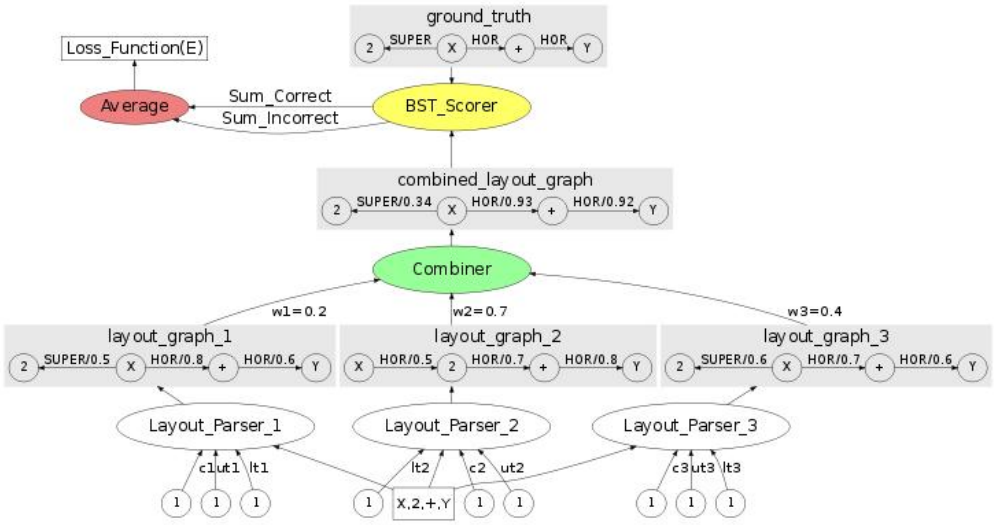


Figure 3.3: GTN Math Parser Combination Architecture

parameters, c which controls the vertical position of symbol centroids, u_t which defines the location of upper regions around symbols (e.g. superscript) and l_t which defines the location of lower regions around symbols(e.g subscript) for different *layout classes* assigned to symbols based on their assigned class.

Individual penalties for each edge on different BSTs are computed using the penalty function described in previous section. After the BSTs with edge penalties have been produced, the BSTs are passed to the *Combiner* module, which has a separate weight for each incoming parser output. The Combiner performs a linear combination of weighted penalties of spatial relationships between two symbols produced by different parsers. This is done by combining all BSTs into a single graph, summing the penalties for any common edges after they have been weighted by their associated connection weight w_i, and then applying Prim's minimum spanning tree algorithm to produce the final BST. For example, consider the $SUPER$ relationship between "X" and "2" in Figure 3.3, where the resulting penalty is

$0.5 * 0.2 + 0.4 * 0.6 = 0.34$. Since penalty is kept between [0,1], weighted penalty is rounded off to 1 if its greater that 1.0.

The Combiner module output is passed to the BST Scorer module which computes the tree edit distance between the combined BST and ground truth. The set of graph edges E is partitioned into three disjoint sets: C, the set of correct edges, I_p, the set of edges with an incorrect parent, and I_r, the set of edges with a correct parent, but an incorrect relationship. Let N be the total number of edges in the combined BST. We define two edge penalty sums, $\sum_C$ for the set of correct edges, and $\sum_{P_r}$ the sum of penalties for I_r.

The function $Error_d$ minimized by the GTN is

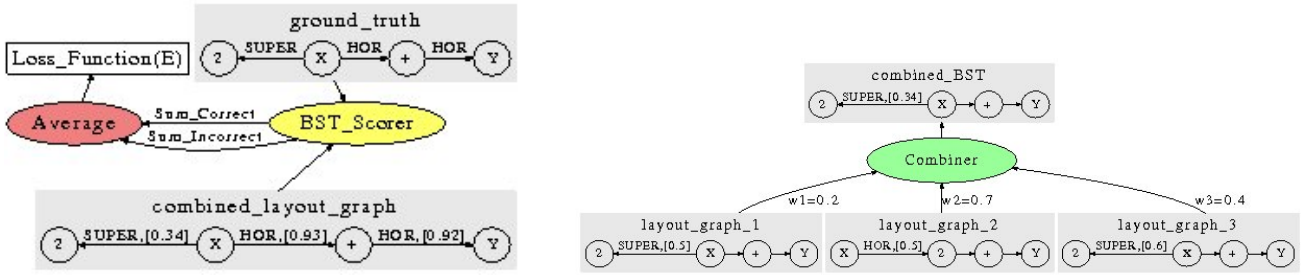
$$Error_d(C, I_p, I_r) = 1 - \frac{(\sum_{I_p} + \sum_{I_r}) + (|C| - \sum_C)}{|C| + |I_p| + |I_r|} = 1 - \frac{(\sum_{I_p} + \sum_{I_r}) + (|C| - \sum_C)}{|E|} \tag{3.11}$$

$Error_d$ produces a loss in $[0, 1]$. This function is design for *disriminative* learning [3], with the goal of maximimizing penalties on incorrect edges, and minimizing penalties on correct edges.

3.2.1 Learning through Backpropagation

All modules in a GTN must be differentiable with respect to their inputs, allowing error gradients to be backpropagated through each module. The modules in our system include the BST Scorer (a sum module), the Combiner module, and one or more parsers producing graphs with edge penalties in $[0, 1]$.

The BST Scorer module is a simple 'average' module that calculates the average



a. Gradients of error at BST Scorer module b. Gradients of error at Linear Combiner

Figure 3.4: Backpropagation in the GTN. Penalties are shown in square brackets

penalty for correct and incorrect edges in the combined BST (see Figure 3.4a). Let E be the set of edges with the BST. Therefore partial derivative for $Error_d$ w.r.t a penalty on combined BST edge is given using Equation (2) as follows.

$$(\frac{\partial Error_d}{\partial e_P}) = (\frac{\partial}{\partial e_P}(1 - \frac{(\sum I_p + \sum_{I_r}) + (|C| - \sum_C)}{|E|})) \tag{3.12}$$

$$\left(\frac{\delta Error_d}{\delta e_P}\right) = \begin{cases} \frac{1}{|E|} & if\ e \in C \\ -\frac{1}{|E|} & if\ e \in I_r \\ -\frac{1}{|E|} & if\ e \in I_p \end{cases}$$

The partial derivatives for $Error_d$ w.r.t a penalty on a correct edge will be a positive and partial derivatives for $Error_d$ w.r.t a penalty on a incorrect edge will be a negative. Such a discriminative learning approach minimizes correct edge penalties and also maximizes penalties on incorrect edges, thus reducing the likelihood of selecting incorrect edges in the Combiner module.

For the Combiner, the input weights w_i are used to scale edge penalties in a graph

containing all incoming edges, from which a minimum penalty spanning tree is computed (see Section 3.2, and Figure 3.4b).

Consider an edge e between symbols X and 2 in the combined BST of Figure 3.4. Let e_P be the relationship penalty (0.34). Let $w_1(0.2)$ be the input weight to the first parser, $w_2(0.7)$ to the second parser and $w_3(0.4)$ to the third parser. Let e_{p1} (0.5), e_{p2} (0.5), and e_{p3} (0.6) be the penalties on the edges $e1$, $e2$ and $e3$ between X and 2 in each of the input BSTs. For each weight w_i, we have:

$$\frac{\partial Error_{d,e}}{\partial w_i} = \frac{\partial Error_d}{\partial e_P}\frac{\partial e_P}{\partial w_i} \tag{3.13}$$

The above equation states that the partial derivative of cumulative error $Error_{d,e}$ w.r.t a input weight w_i is equal to product of partial derivative of cumulative error $Error_{d,e}$ w.r.t an edge penalty on combined BST and partial derivative of that same edge penalty w.r.t input weight w_i. Here we are just applying the chain rule of computing partial derivatives.

Similarly, for each edge penalty e_{pi} in one of the original BSTs we have:

$$\frac{\partial Error_{d,e}}{\partial e_{pi}} = \frac{\partial Error_d}{\partial e_P}\frac{\partial e_P}{\partial e_{pi}} \tag{3.14}$$

The above equation states that the partial derivative of cumulative error $Error_{d,e}$ w.r.t an edge penalty e_{pi} on an input BST is equal to product of partial derivative of cumulative error $Error_{d,e}$ w.r.t an edge penalty on combined BST and partial derivative of that same edge penalty w.r.t edge penalty e_{pi}.

Prim's algorithm iteratively finds the least expensive (minimum penalty) edge between two symbols in input BSTs and adds it to the minimum spanning tree. If edge ei

contributed in linear combination to produce penalty for e i.e if $ei \in e$, then,

$$\left(\frac{\partial e_P}{\partial w_i}\right) = \begin{cases} e_{pi} & if\ ei \in e \\ 0 & if\ ei \notin e_i \end{cases}$$

Similarly,

$$\left(\frac{\partial e_P}{\partial e_{pi}}\right) = \begin{cases} w_i & if\ ei \in e \\ 0 & if\ ei \notin e \end{cases}$$

Substituting value of $(\frac{\partial Error_d}{\partial e_P})$ in equations 3.13 and 3.14, we get,

$$\left(\frac{\partial Error_{d,e}}{\partial w_i}\right) = \begin{cases} \frac{1}{|E|}\frac{\partial e_P}{\partial w_i} & if\ e \in C \\ -\frac{1}{|E|}\frac{\partial e_P}{\partial w_i} & if\ e \in I_r \\ 0 & if\ e \in I_p \end{cases}$$

$$\left(\frac{\partial Error_{d,e}}{\partial e_{pi}}\right) = \begin{cases} \frac{1}{|E|}\frac{\partial e_P}{\partial e_{pi}} & if\ e \in C \\ -\frac{1}{|E|}\frac{\partial e_P}{\partial e_{pi}} & if\ e \in I_r \\ 0 & if\ e \in I_p \end{cases}$$

Given the set of such edges E in the combined BST, the partial derivative of the error w.r.t. w_i is is obtained by computing the gradient of error for each edge in the combined BST as above, and then summing the derivatives:

$$\frac{\partial Error_d}{\partial w_i} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial w_i} \tag{3.15}$$

The Linear Combiner module's weights w_i are updated as in the following:

$$w_i(n+1) = w_i(n) - \eta * \frac{\partial Error_d}{\partial w_i} \tag{3.16}$$

where η is the learning rate.

In order to compute min at switching surfaces (when edges have same penalty values) we compute penalties using the same penalty functions at a very small higher interval and select minimum of newly computed penalty values. This ensures that we select correct(minimum) penalty function each time there is a tie. We also know that min function is continuous and reasonably regular to achieve convergence using gradient-based learning algorithm [3].

Next we define gradient of $Error_{d,e}$ w.r.t scaling factor (b) of a penalty function.

$$\frac{\partial Error_{d,e}}{\partial b} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{\partial e_{pi}}{\partial b} \tag{3.17}$$

From Equation 3.1, we have,

$$\frac{\partial e_{pi}}{\partial b} = \frac{\partial(\frac{1}{1+e^{-by_\Delta}})}{\partial b} = \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial b} \tag{3.18}$$

Now depending upon the experimental setup, we have single penalty function for all combining parsers or individual penalty function for each combining parser (described in detail in section 3.2.2).

If we have single penalty setup, then for each edge on each combining BST, we take the penalty(p_i) associated with that edge and using the above equation we compute the

partial derivatives of $Error_{d,e}$ w.r.t b. The total gradient w.r.t b will be the sum of all such partial derivatives computed.

$$\frac{\partial Error_d}{\partial b} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial b} \tag{3.19}$$

However in individual penalty setup, each parser in the setup is associated with individual penalty function. For each edge on a combining BST, we compute the partial derivative of $Error_{d,e}$ w.r.t to that BST's associated scaling factor b_i using equation 3.17. The total gradient w.r.t scaling factor b_i is equal to the sum of all such partial derivatives computed for all edges of that BST.

$$\frac{\partial Error_d}{\partial b_i} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial b_i} \tag{3.20}$$

Similarly we compute total gradient of $Error_{d,e}$ w.r.t to scaling factors of other combining BSTs individually.

The penalty functions' scaling factor are updated as in the following:

$$b_i(n+1) = b_i(n) - \eta * \frac{\partial Error_d}{\partial b_i} \tag{3.21}$$

where η is the learning rate.

The error gradients for each of the parsers need to be computed relative to the parameters defining the vertical locations of $SUPER/HOR$, $ABOVE/CONTAINS$ and $UPPER/CONTAINS$ regions (ut) and $SUBSC/HOR$, $BELOW/CONTAINS$ and $LOWER/CONTA$

regions (lt), and the vertical position of symbol centroids (c). Both are defined as a percentage of bounding box height. The partial derivatives of $Error_d$ w.r.t region parameters t and c are calculated as:

$$\frac{\partial Error_{d,e}}{\partial ut} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{\partial e_{pi}}{\partial ut} \tag{3.22}$$

$$\frac{\partial Error_{d,e}}{\partial lt} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{\partial e_{pi}}{\partial lt} \tag{3.23}$$

$$\frac{\partial Error_{d,e}}{\partial c} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{\partial e_{pi}}{\partial c} \tag{3.24}$$

The relationship penalty e_{pi} is based on the proximity of the neighboring (child) symbol's centroid C to the region threshold T and is calculated using equation 3.1 explained in previous section.

From Equation 3.1, we have,

$$\frac{\partial e_{pi}}{\partial T} = \frac{\partial(\frac{1}{1+e^{-by_\Delta}})}{\partial T} = \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial T} \tag{3.25}$$

Similarly,

$$\frac{\partial e_{pi}}{\partial C} = \frac{\partial(\frac{1}{1+e^{-by_\Delta}})}{\partial C} = \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial C} \tag{3.26}$$

This region threshold T is in turn defined by threshold parameter t and symbol's centroid C is in turn defined by centroid parameter c. This is shown in Figure 3.5.

Symbol Class	y-centroid	Thresholds			
		BELOW	ABOVE	SUBSC	SUPER
Non-Scripted unary/binary operators and relations (+,-,=,≥,→,etc.)	$\frac{1}{2}H$	$\frac{1}{2}H$	$\frac{1}{2}H$	–	–
Open Bracket (, {, [	cH	$minY$	$maxY$	–	–
Root ($\sqrt{}$)	cH	$minY$	$maxY$	tH	$H-(tH)$
Variable Range $\Sigma, \int, \Pi, \cup, \cap$	$\frac{1}{2}H$	tH	$H-(tH)$	tH	$H-(tH)$
Plain: Ascender 0...9, A...Z, b,d,f,h,i,k,l,t, $\Gamma, \Delta, \Theta, \Lambda, \Xi, \Phi, \Psi, \Omega, \delta, \theta, \lambda$	cH	tH	$H-(tH)$	tH	$H-(tH)$
Plain: Descender g,p,q,y, $\gamma, \eta, \mu, \rho, \chi, \psi$	$H-(cH)$	$\frac{1}{2}H+t\frac{1}{2}H$	$H-t\frac{1}{2}H$	$\frac{1}{2}H+t\frac{1}{2}H$	$H-t\frac{1}{2}H$
Plain: Centered All other symbols (including Close Brackets)	$\frac{1}{2}H$	tH	$H-(tH)$	tH	$H-(tH)$

Figure 3.5: Threshold and Centroid Table [28]

Again applying chain rule we get:

$$\frac{\partial e_{pi}}{\partial ut} = \frac{\partial e_{pi}}{\partial T}\frac{\partial T}{\partial ut} \tag{3.27}$$

$$\frac{\partial e_{pi}}{\partial lt} = \frac{\partial e_{pi}}{\partial T}\frac{\partial T}{\partial lt} \tag{3.28}$$

$$\frac{\partial e_{pi}}{\partial c} = \frac{\partial e_{pi}}{\partial C}\frac{\partial C}{\partial c} \tag{3.29}$$

Substituting $\frac{\partial e_{pi}}{\partial ut}$, $\frac{\partial e_{pi}}{\partial lt}$ and $\frac{\partial e_{pi}}{\partial c}$ in Equation 3.22, 3.23 and 3.24 respectively, we have,

$$\frac{\partial Error_{d,e}}{\partial ut} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial T} \frac{\partial T}{\partial ut} \tag{3.30}$$

$$\frac{\partial Error_{d,e}}{\partial lt} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial T} \frac{\partial T}{\partial lt} \tag{3.31}$$

$$\frac{\partial Error_{d,e}}{\partial c} = \frac{\partial Error_{d,e}}{\partial e_{pi}} \frac{e^{-by_\Delta}}{(1+e^{-by_\Delta})^2} \frac{\partial(-by_\Delta)}{\partial C} \frac{\partial C}{\partial c} \tag{3.32}$$

We compute $\frac{\partial Error_d}{ut}$ and $\frac{\partial Error_d}{lt}$ by summing the gradient of error for each each edge penalty in set of edges E as follows,

$$\frac{\partial Error_d}{\partial ut} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial ut} \tag{3.33}$$

$$\frac{\partial Error_d}{\partial lt} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial lt} \tag{3.34}$$

Similarly we compute $\frac{\partial Error_d}{c}$ by summing the gradient of error for each each edge penalty in set of edges E as follows,

$$\frac{\partial Error_d}{\partial c} = \sum_{e \in E} \frac{\partial Error_{d,e}}{\partial c} \tag{3.35}$$

Each parser's ut, lt and c parameter values are updated as in the following:

$$ut(n+1) = ut(n) - \rho * \frac{\partial Error_d}{\partial ut} \tag{3.36}$$

$$lt(n+1) = lt(n) - \rho * \frac{\partial Error_d}{\partial lt} \tag{3.37}$$

$$c(n+1) = c(n) - \rho * \frac{\partial Error_d}{\partial c} \tag{3.38}$$

where ρ is the learning rate.

3.2.2 Experiment

Experiments are based on INFTY dataset [23] which consists of ground truth database for characters, words and mathematical expressions. There are 21,056 mathematical expressions present in the INFTY dataset which are preprocessed to produce a dataset to be used for our experiment. Experiment consists of dividing the dataset into different sets for GTN training, validation and testing. Experiments are mainly divided into two sections as explained in the following subsections. Each experiments are run multiple times and results are recorded for each run and analyzed.

3.2.2.1 Data Set

Experimental dataset consists of INFTY dataset [23] that consists of 21,056 mathematical expressions. Dataset is divided into two parts: one that contains text data in Microsoft Access format and other that contains image data in the form of PNG files. Text data consist of mathematical symbols described using 29 attributes. Using these 29 attributes, individual mathematical expressions are extracted into separate expressions as given in image data. These expressions are then preprocessed in the following ways:

- Removing Matrices: DRACULAE expression grammar parses a subset of dialects of mathematics and matrices are not included in the subset. Hence all the expression in the dataset containing matrices are removed in the resulting dataset.

- Removing Malformed: All those expressions that contain more than 1 symbol (more than start symbol) having -1 or null parent, in otherwords if there are more than 1 symbol in an expression having no parent then that expression is to be malformed expression and hence removed the the resulting dataset.

- Removing Single-Symbol: As GTN learning is based on learning relationships between two symbols in an expression, all expressions containing 1 symbol are removed from the resulting dataset as they do not contribute anything in GTN learning process.

- Shuffling: After all the above preprocessing, the expressions are shuffled inorder to randomize expressions in different sets. The order in which data is presented for training is important and its is believed that having randomized data improves the accuracy during training as shuffling enables data to be as far as possible thus making the set statistically unbiased.

After preprocessing, 14,687 mathematical expressions are extracted into the resulting dataset out of 21,056 expressions. These 14,687 expressions are later divided in different sets for training, validation and testing. Out of 14,687 mathematical expressions, 8814 expressions are included in training set, 3107 expressions are included in validation set and 2766 expressions are included in the test set. A separate validation set is used to compare the performance of the networks by evaluating the error function against this set, and the network having the smallest error with respect to the validation set is selected.

Experimental setup is mainly divided into two parts where we try to analyze the effect of having one penalty function for all DRACULAE parsers used in GTN and having individual penalty functions for each DRACULAE parser used in GTN.

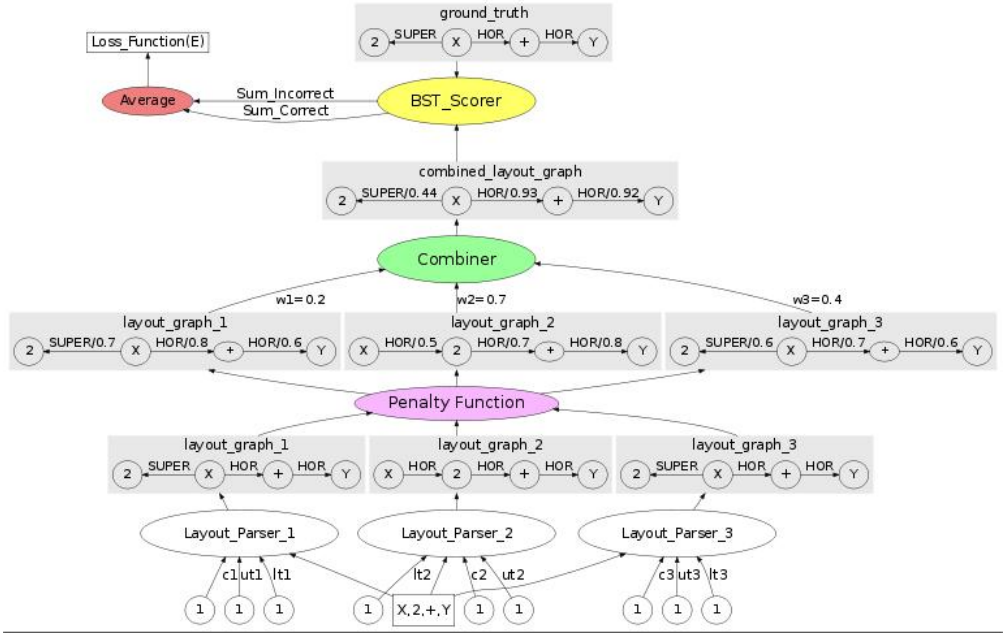


Figure 3.6: GTN-Single Penalty Architecture

3.2.2.2 Single Penalty Function Setup

In this setup a single penalty function is used to compute a penalty on each edge of the BST as described in section 3.1. The setup is shown in Figure 3.6. Penalty for each edge in each BST is computed and the resulting BSTs are fed as into to Combiner module for weighted combination of penalized edges to produce combined BST. During backpropagation learning, gradients computed w.r.t to each penalized edge on each BST are combined to update the scaling factor of penalty function.

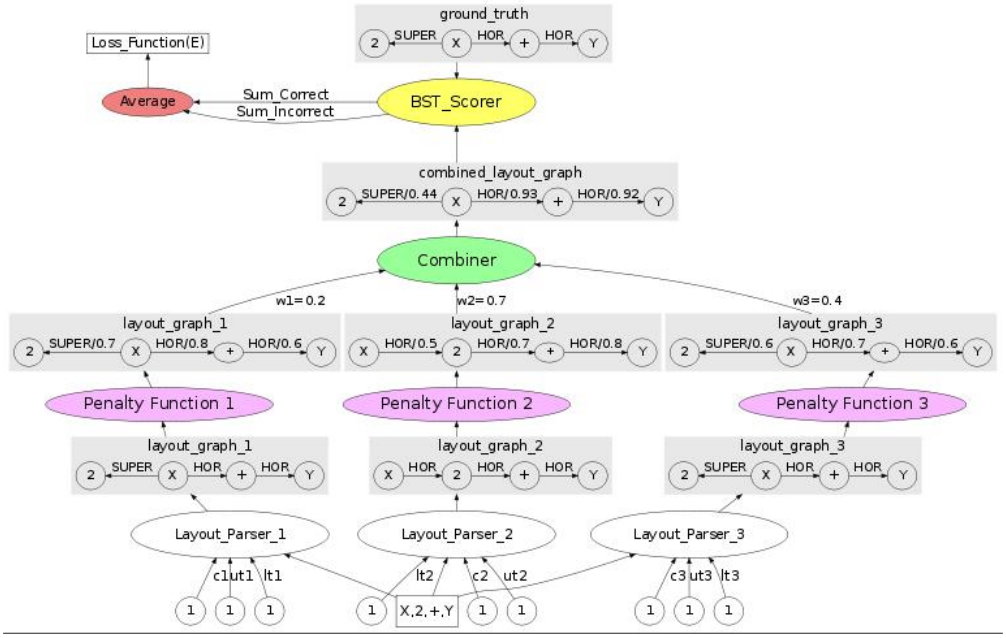


Figure 3.7: GTN-Individual Penalties Architecture

3.2.2.3 Multiple penalty functions Setup

In this setup each BST from DRACULAE parser are penalized with individual penalty function as shown in Figure 3.7. Edge penalties are computed using separate penalty functions and then the resulting BSTs are fed into combiner module for weighted combination of penalized edges. During backpropagation learning, gradients computed from a BST's edges are combined to update the scaling factor of penalty function used to penalized the BST.

3.2.3 Training and testing

We assess the recognition performance of GTN on Suzuki's INFTY dataset using 3 metrices, namely 1) the ratio of correctly recognized mathematical expressions to total

number of expressions used in test set 2) total number of incorrectly recognized symbol's parent (Parent Error) in all the expressions used in test set and 3) total number of incorrectly recognized symbol's relationship with its parent symbol (Relationship Error).

We adopted batch method of learning where GTN is trained with all the expressions in training set (called as epoch) and after each epoch all the GTN parameters are updated. After each epoch, the GTN network perfomance is evaluated against validation set and GTN network with minimum validation error is selected. Once the minimum validation error GTN network is selected, the test set is run against the learned GTN network and the resulting combined BST for each expression is compared against the grund truth and a tree edit distance is computed between ground truth and combined BST as explained in section 2.6. Edit distance of 0 means the expression is correctly recognized by the GTN network and count of correctly recognized expression is increased. On the other hand, if the edit distance is more than 0, corresponding parent error and relationship error are evaluated and noted. Same data sets are used for two different setups explained above and similar processes of training and testing are performed and the results are noted. Since the experiment used gradient descent method of learning, there is a chance of encountering local minima instead of global minima. To try to avoid local minima problem, each experiment is run multiple times.

Chapter 4

Results and Discussion

The results from the experiments described in Chapter 3 are presented in this chapter. Section (4.1) presents the experimental results from each individual GTN setup and provides the expression rate along with averages and standard deviations. Section (4.2) provides a distribution of different error types for each GTN setup. Section (4.3) presents a compariative error analysis from different experimental GTN setup for different error types.

4.1 Expression Rate

Following Figures 4.1, 4.2 provide the expression rates from each individual GTN experimental setup which include 1 parser, 2 parsers and 3 parsers. The graphs are bar charts in which each bar represents an expression rate averaged over 10 experimental runs for a given GTN setup. The graphs also show the standard error in terms of a vertical line at the top of each bar. The standard error shows the dispersion of recognition rates around its average for each GTN setup.

4.1.1 Single Penalty Scale

In this experimental setup we use a single vertical scaling factor to define our penalty function as described in 3.1. In such an experiment since we are using one scaling factor,

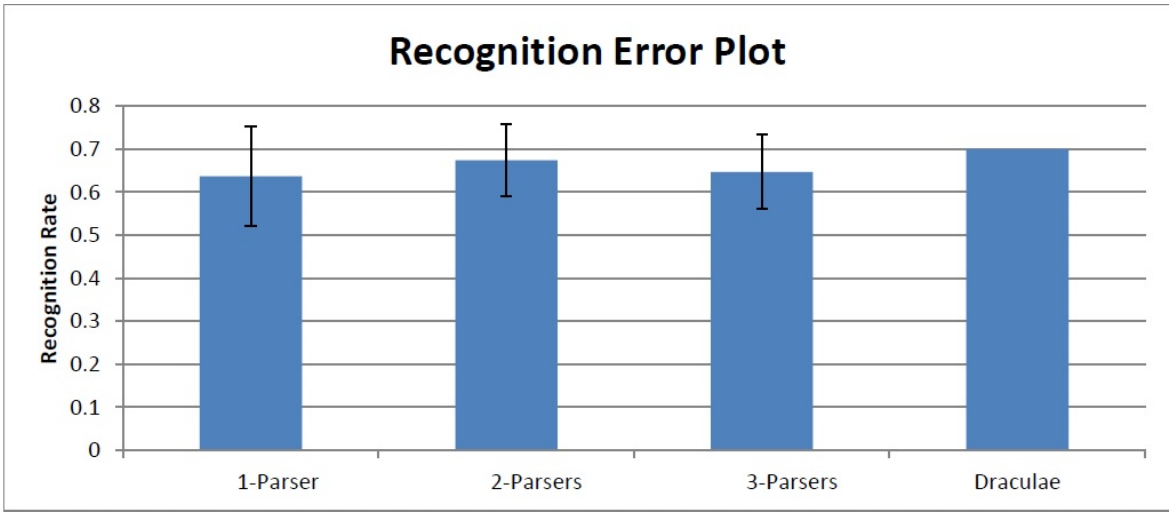


Figure 4.1: Average recognition rates along with standard errors for single-scale experiment

learning of both upper and lower thresholds affects the learning of scaling factor.

4.1.2 Discussion

As can be inferred from the graph 4.1, in all the GTN setups we get considerable good recognition rates higher than the recognition rate from plain DRACULAE. For GTN structure with 1 parser shows maximum recognition rate of 74% as compared to 70% of DRACULAE run but with higher standard error as compared to other GTN structures. This is due to few outliers that have low recognition rates for some runs. This is contributed by a phenomenon called local minima problem associated with the experiments using gradient descent method of learning. Local minimum problem occurs when backpropogation learning converge at local minimum value in the search space instead of global desired minimum. It can be a result of complex error function having multiple learning parameters or lack of sufficient learning data sets.

2-parsers GTN structure, which involves combination of 2 DRACULAE parsers in a GTN setup, also achieves a maximum recognition rate of 74% which is higher than plain DRACULAE and have smaller standard error rate as compared to other GTN structures. Smaller the standard error, smaller is the variation from the average and a better representation of recognition result. Thirdly there is 3-parsers GTN setup which achives a maximum recognition rate of 73% and with a standard error in between 1-parser and 2-parsers GTN structures. This graph, overall, shows that with the such a experiment which involves combination of parsers we can acheive better recognition rates however results obtained were not as consistent as we desired.

As mentioned above some of the experimental runs have witnessed local minima problem associated with gradient descent learning. By employing other means and methods focussed to reduce local minima problem we can achieve more consistent recognition results with smaller standard errors. There are other factors that can be attributed to lower performance one of which can be explained by contrasting our experiment with Lecun's experiment. In Lecuns's experiment there is always a particular interpretation among combining interpretations that is always correct. This advantage allows Lecun's to obtain learning at the 'string' level(instead of single character level). This we can say because they are making use of combined penalty of one single interpretation(a correct one offcourse). However since we dont have a correctly recognized interpretation(expression) we are still performing the learning at the individual relationship level (instead of whole mathematical expression level). This we can say as we are looking into single relationship correctness or incorrectness. This also allows their final error function to fit well for their combining architecture, as it tries to reduce the penalty of correct character sequence as a whole and

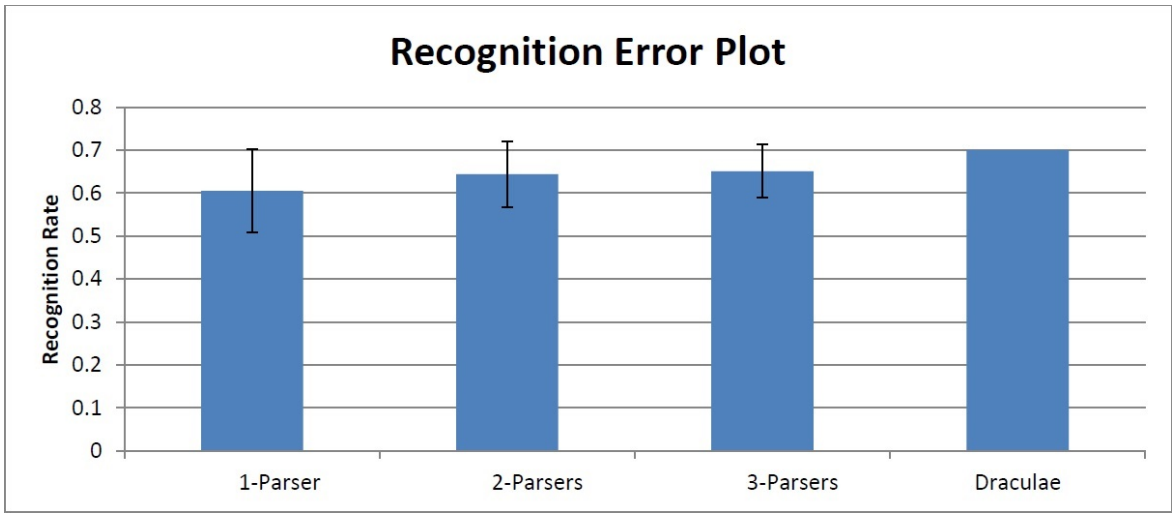


Figure 4.2: Average recognition rates along with standard errors for two-scales experiment

increase the penalty of incorrect character sequence as a whole.

Other performance limiting factor can be attibuted to the more deterministic nature of DRACULAE parsers which makes learning harder. DRACULAE consists of multiple heuristic rules that cannot be used in learning especially in gradient based learning.

4.1.3 Upper and Lower Penalty Scales

In this experiment we split the vertical scaling factor used in penalty function into two: upper scale to control/scale penalty for upper regions and lower scale to control/scale the penalty for lower regions. By dividing the scaling factor into two we are ensuring that the upper and lower regions' threshold learnings are not affected by each other. So for each scale, gradients are computed independently and propagated back to compute the gradients of upper and lower thresholds independent of each other. Figure 4.2 shows the expression recognition rate plot from such an experiment.

4.1.4 Discussion

AS can be seen from the graph, the bars looks more or less similar to what we obtained for single scale experimental results. We see lower maximum recognition rate reached with this experiment as compared to single penalty scale experiment. However this experimental results shows lower error rates as compared to single scale experiment as the number of parsers in GTN increases. This means with separate scaling factors we are getting less variation in the recognition rate with increasing number of parsers in GTN which means we are getting a better representation of recognition result.

4.2 Relation and Parent Error

As mentioned in 3.2 section, the recognition error is computed in terms of three disjoint sets from GTN - set of correctly recognized edges, set of edges with an incorrect parent (parent error) and set with edges with a correct parent but an incorrect relationship (relation error). Error function is given at 3.11. Following graphs give the distribution of relation and parent errors from different GTN structures. These graphs provide an overview of how and what distribution of these types of error affect the resulting recognition rate.

4.2.1 Discussion

Figure 4.3 shows a column bar chart which gives the relation and parent error distribution for 10 different runs for 1-ParserGTN structure. The graph also shows the recognition rate at the top of each column for that particular run. An obvious inference can be made from the graph is that lower the total error, higher is the recognition rate. As can be seen from the graph parent errors are higher contributing elements than relation errors in each

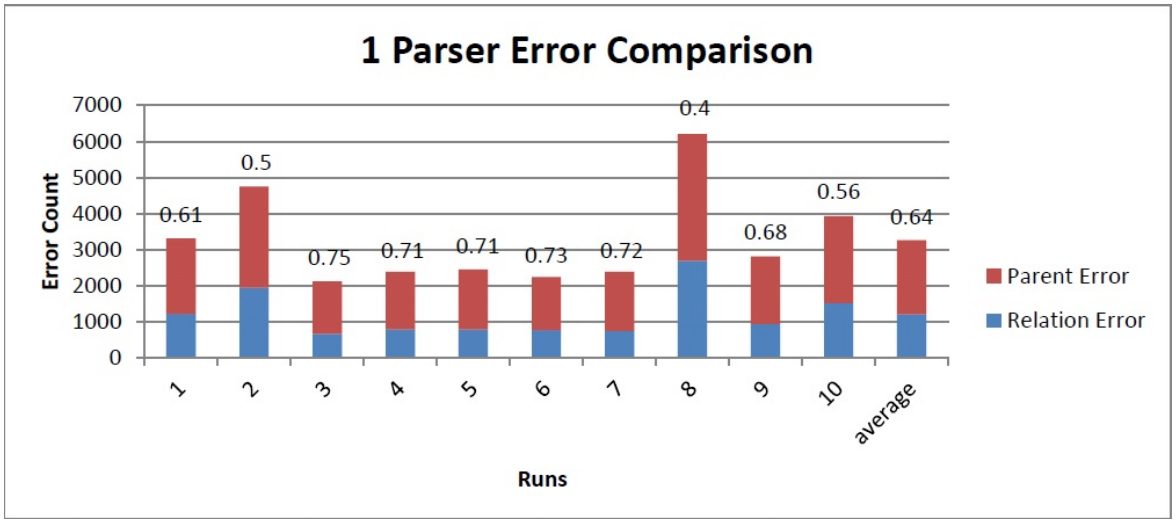


Figure 4.3: Segmented bar chart showing Relation and Parent error distribution of 10 runs for 1-Parser GTN structure. Red represents parent error count and blue represents relation error count. Corresponding recognition rate for the run is also shown at the top of each bar

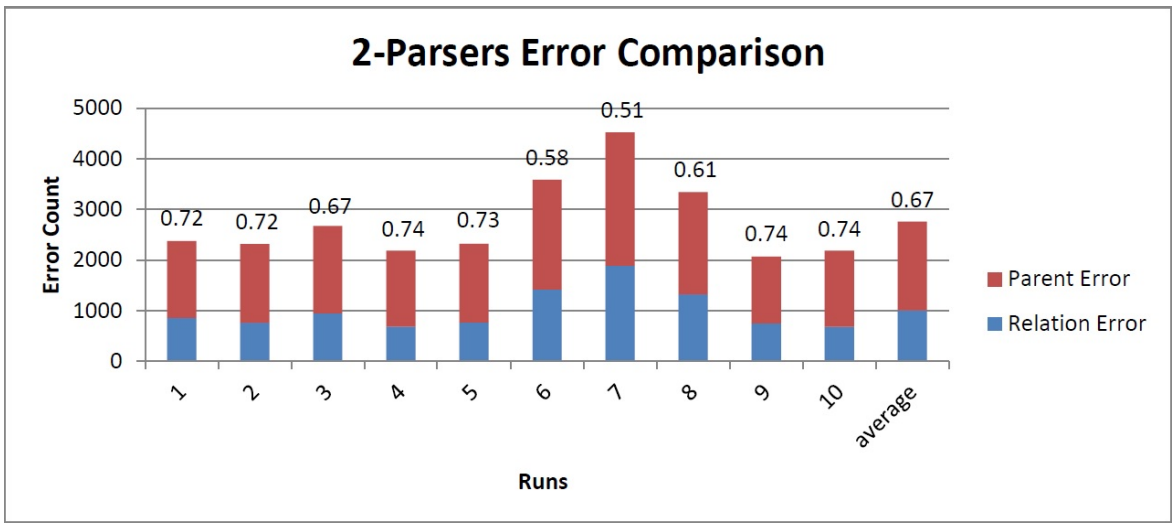


Figure 4.4: Segmented bar chart showing Relation and Parent error distribution of 10 runs for 2-Parser GTN structure. Red represents parent error count and blue represents relation error count. Corresponding recognition rate for the run is also shown at the top of each bar

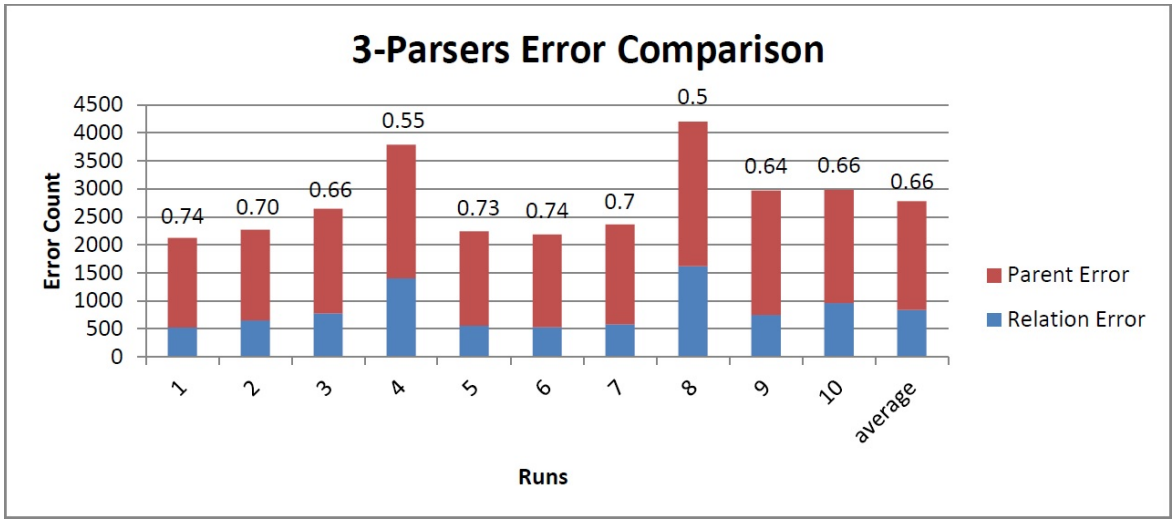


Figure 4.5: Segmented bar chart showing Relation and Parent error distribution of 10 runs for 3-Parser GTN structure. Red represents parent error count and blue represents relation error count. Corresponding recognition rate for the run is also shown at the top of each bar

run. Therefore for different expressions used in the experiment there are more instances of edges in final BST with incorrectly recognized parents than edges with incorrectly recognized relationships with parent. From the graph it is also visible that for the runs with minimum parent and relation errors has better recognition rate. Figures 4.4 and 4.5 similarly shows the error distribution for 10 different runs of 2-parsers and 3-parsers GTN structures respectively.These graphs also shows the same behavior in which we have higher parent errors as compared to relation errors. Each of the run showing more parent errors as compared to relation errors can be explained using the following two figures:

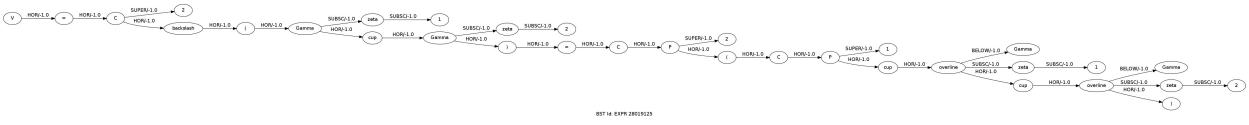


Figure 4.6: Ground Truth BST

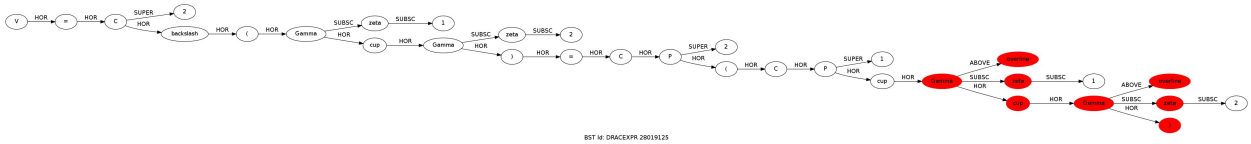


Figure 4.7: DRACULAE Parsed BST

Figure 4.6 represents the ground truth BST for one such expression from INFTY dataset and Figure 4.7 represents DRACULAE parsed BST. As can be seens from the figure 4.6 symbol Overline is HOR to symbol Cup and symbols Gamma, Zeta and another Cup are BELOW, SUBSC and HOR to Overline respectively. However in resulting parsed BST (Figure 4.7) we found that symbol Gamma was incorrectly recognized as HOR to symbol Cup which caused Symbols Overline, Zeta and Cup to be incorrectly associated as child symbols of Gamma. This shows a single misrecognition of association between symbols (Cup and Gamma here) leads to 3 parent errors and in our experiments when we obtain a parent error we avoid an additional count of relation error(if there is one) to wrong parent. This is essential to avoid incorrect count of errors. In another term we avoid associating error count of 2 for incorrect parent and relation error just because had the parent was correctly recognized, the relationship with the correct parent could have been correct. As a result we end up having more parent errors than relation errors.

4.3 Error Comparison

Following figures shows the errors based comparison between different GTN structures. Figure 4.8 shows a bar chart depicting parent error comparison between GTNs with 1, 2 and 3 parsers. The bars charts gives the min, average and max view of parent error from different GTN structures. Similarly, Figure 4.9 shows a bar chart depicting relation

error comparison between different GTN structures.

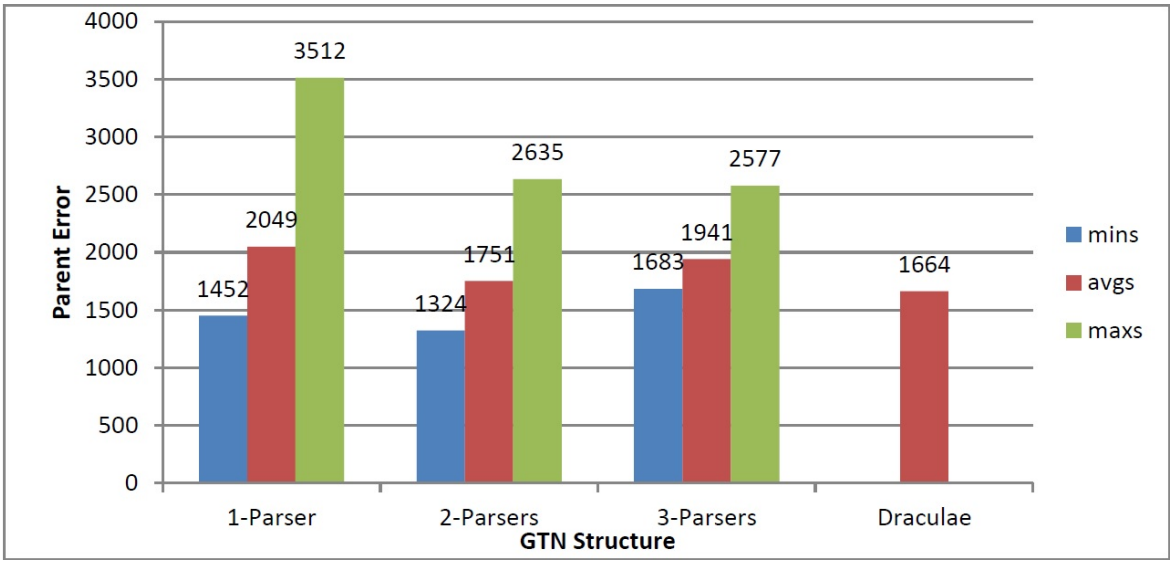


Figure 4.8: Parent errors comparison between GTN structures having 1, 2 and 3 DRACULAE parsers involved

4.3.1 Discussion

As can be inferred from the parent error graph 4.8, 2-Parsers GTN has average parent errors more or less close to DRACULAE parent errors and hence gives better result as compared to other 2 GTN structures. 2-Parser GTN also have loweest minimum parent errors as compared to other 2 GTN structures. From graph 4.9, it can be seen that 3-parsers GTN gives lowest average relation errors and is very close to relation errors from DRACULAE. 2-Parsers GTN also shows the lowest min and max relation errors as compared to other GTN structures thus resulting into better recognition rate. However to reitrerate, as we have seen that parent errors contribute highly in total recognition error and therefore any GTN structure which results into lower parent error will produce better recognition

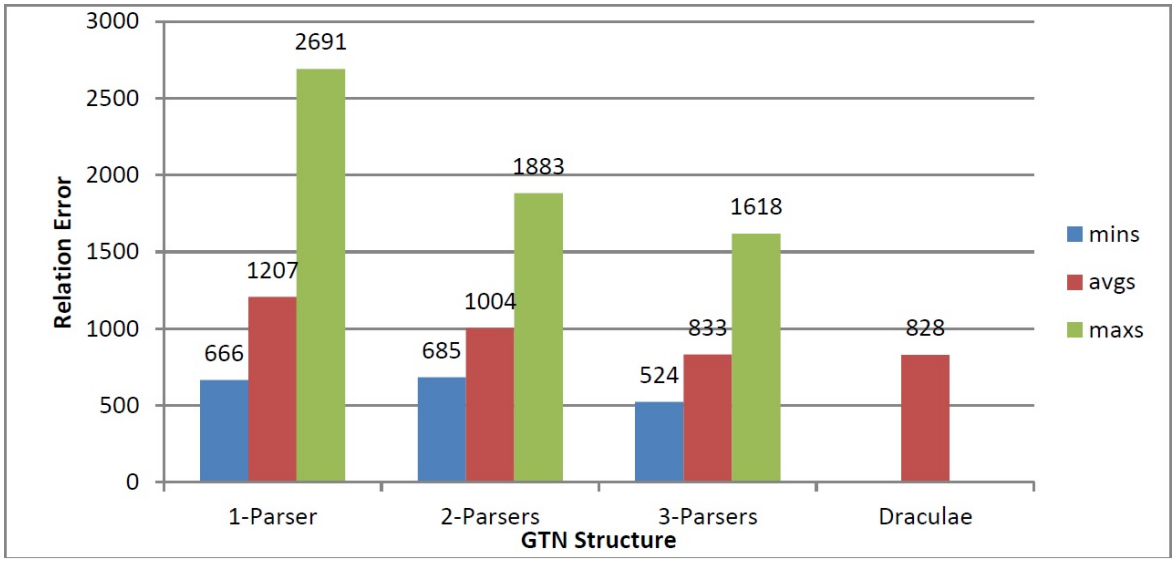


Figure 4.9: Relation errors comparison between GTN structures having 1, 2 and 3 DRACULAE parsers involved

results and vice-versa.

4.4 Summary

Our experiment employs a novel method to combine multiple structural recognition parsers (mathematical recognition DRACULAE parsers [28]) inorder to obtain higher recognition rate. Our experiments support our hypothesis that by combining the recognition results from multiple similar structural recognition parsers, we can improve structural recognition and obtain recognition rate higher than individual parser's recognition rate. From the above results we can see that by combining multiple DRACULAE parsers in GTN we can obtain a recognition rate of maximum 74% which is higher than the best result from vanilla DRACULAE result which is nearly 70%. Results also shows that the number

of parent and relation errors can also be reduced from the combination. The best results in Figure 4.8 shows a reduction of 20% in parent errors for 2-Parser BST and around 13% reduction in parent errors for 1-Parser BST. 3-Parsers BST shows a slight increase of 1% in parent errors. The best results in Figure 4.9 shows a 37% reduction in relation errors for 3-Parsers BST, 20% reduction for 1-Parser BST and 17% reduction in 2-Parsers BST. The average results doesnt look as promising due to low recognition in some experiment runs affecting the overall average. Above results need to be improved inorder to maximize average recognition rates and obtain consistent result.

Chapter 5

Conclusion

Thesis Statement: Recognition performance of stuctural analysis parsers like mathematical recognition parsers can be improved by adopting intelligent parser combination method.

We have presented a machine learning algorithm of combining multiple structural recognition algorithms thats supports our hypothesis. Similar experiment was presented by LeCun et. al [3], but it was applied to one dimensional document symbol recognition problem. We showed, in our experiments, that we can combine multiple instances of 2-dimensional mathematical recognition DRACULAE parsers to obtain a maximum recognition rate of 74% as compared to the best rate of 70% from plain DRACULAE. Our experiment also showed that the total parent and relation errors can be reduced with the combination leading to better recognition of symbol's parent and its spatial relationship with its parent.

The average results were not as promising due to lower recognition results from some experimental runs which lower the average result. Other learning methods can be employed and tested in order to maximize average recognition rates and obtain consistent result. Section 4.2 explained that misrecognition of parent symbol is a higher contributing

factor in recognition error, therefore we need to employ better handling of parent errors in our experiment to reduce the number of parent errors (may be by highly penalizing parent errors or using an error function that weights parent errors higher) and thus we can obtain better recognition rate than what we obtained from the above experiments.

5.1 Future Work

The performance of our method can be improved by using other performance improving machine learning algorithms like Boosting. Most widely used boosting algorithm is AdaBoost (AdaptiveBoosting). AdaBoost train each of the base classifier in sequence using a weighted form of the data set in which weighting coefficient of each data point are adjusted according to the performance of the previous trained classifier so as to give greater weight to misclassified data points. Boosting technique like AdaBoost can be applied to GTN structure to either boost the intermediate results generated by a module of GTN or boost the result given by entire GTN structure.

Our learning technique can be applied to other structural recognition algorithms like table recognition [30] where we combine the results from multiple different algorithms. Such a technique can overcome the recognition limits of each algorithm used in combination and combined output can take advantage of each algorithm's strength and reinforce it if the algorithmic parameters are tuned correctly.

Bibliography

[1] Kamal M. Ali and Michael J. Pazzani. Error reduction through learning multiple descriptions. *Mach. Learn.*, 24(3):173–202, 1996.

[2] N. Arica and F.T. Yarman-Vural. Optical Character Recognition for Cursive Handwriting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 801–813, 2002.

[3] Y. Bengio, Y. Lecun, L. Bottou, and P Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] D. Blostein and A. Grbavec. Recognition of Mathematical Notation. *Handbook of Character Recognition and Document Image Analysis*, pages 557–582, 2001.

[5] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. pages 152–160, 1998.

[6] Eric Brill and Jun Wu. Classifier combination for improved lexical disambiguation. pages 191–195, 1998.

[7] Brunet-Manquat and Francis. Syntactic Parser Combination For Improved Dependency Analysis. *Workshop On Robust Methods In Analysis Of Natural Language Data ROMAND*, 2004.

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

[9] Y.K. Chen and J.F. Wang. Segmentation of Single-or Multiple-Touching Handwritten Numeral String Using Background and Foreground Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1304–1317, 2000.

[10] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. *In Proceedings of the SPIE Visual Communications and Image Processing IV*, 1199:852–863, November 1989.

[11] J. Fiscus. A post-processing system to yield reduced word error rates: Recogniser output voting error reduction (rover). pages 347–352, 1997.

[12] Victoria Fossum and Kevin Knight. Combining constituent parsers. In *HLT-NAACL (Short Papers)*, pages 253–256, 2009.

[13] U. Garain and BB Chaudhuri. Recognition of Online Handwritten Mathematical Expressions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(6):2366–2376, 2004.

[14] U. Garain and BB Chaudhuri. Recognition of Online Handwritten Mathematical Expressions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(6):2366–2376, 2004.

[15] A. Grbavec and D. Blostein. Mathematics recognition using graph rewriting. page 417, 1995.

[16] K. Gyeonghwan and V Govindraju. A Lexicon Driven Approach to Handwritten Word Recognition for Real-Time Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 366–379, 1997.

[17] John C Henderson and Eric Brill. Exploiting Diversity in Natural Language Processing: Combining Parsers. *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing*, pages 184–194, 1999.

[18] Zhi-Qiang Liu Jinhai Cai. Integration of Structural and Statistical Information for Unconstrained Handwritten Numeral Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3):263–270, 1999.

[19] Xue-Dong Tian Ming-Hu Ha and Na Li. Structural Analysis of printed mathematical expressions based on combined strategy. *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian*, pages 13–16, 2006.

[20] M Okamoto and B Miao. Recognition of mathematical expressions by using the layout structure of symbols.

[21] Eric Lecolinet Richard G. Casey. A Survey of Methods and Strategies in Character Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.

[22] Kenji Sagae and Alon Lavie. Parser Combination by Reparsing. *Language Technologies Institute. Carnegie Mellon University. Pittsburgh, PA 15213*, 2006.

[23] M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. In *Proceedings of 8th International Conference on Document*

Analysis and Recognition (ICDAR 2005), volume 2, pages 675–679, Seoul, Korea, 2005. IEEE Computer Society Press.

[24] K.C. Tai. The tree-to-tree correction problem. *Journal of the association for computing machinery*, 26:422–433, 1979.

[25] Ernesto Tapia and Raúl Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In *GREC*, volume 3088, pages 329–340, 2003.

[26] HM Twaakyondo and M. Okamoto. Structure analysis and recognition of mathematical expressions. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, 1995.

[27] Hans Van Halteren, Jakub Zavrel, and Walter Daelemans. Improving data driven wordclass tagging by system combination. pages 491–497, 1998.

[28] Blostein D. Zanibbi, R. and J.R Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002.

[29] R. Zanibbi, A. Pillay, H. Mouchere, C. Viard-Gaudin, and D. Blostein. Stroke-based performance metrics for handwritten mathematical expressions. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 334–338, Sept 2011.

[30] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Decision-based specification and comparison of table recognition algorithms. In *Machine Learning in Document*

Analysis and Recognition, volume 90 of *Studies in Computational Intelligence*, pages 71–103. Springer, 2008.

Vita

Amit Arun Pillay was born in Nagpur, Maharashtra, India on September 21, 1984, the son of Arun Pillay and Uma Pillay. He received the Bachelor of Engineering(B.E) degree in Information Technology from Veermata Jijabai Technological Institute, Mumbai, India in 2006. He obtained his Master of Science (M.S) degree from Rochester Institute of Technology, United States of America. Currently he is working as a Software Engineer at Silverpop, Atlanta, Georgia, US. His research interest includes Machine Learning, Pattern Recognition, Computer Vision and Image Processing. His current research includes combining syntactical pattern recognition techniques.

Permanent address: 615, Shadowood Pkwy SE,
Atlanta, Georgia, US

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.