Generating Synthetic Online Handwritten Mathematical Expressions from Markup Languages

by

Vincenzo Heska

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this essay. This is a true copy of the essay, including any required final revisions, as accepted by my examiners.

I understand that my essay may be made electronically available to the public.

Abstract

We report on Handwritten Math Generator, a software system for generating synthetic online handwritten mathematical expressions to assist in the training of a deep learning recognizer. Our synthesizer is able to create many versions of the same expression, as well as create new expressions from a markup language such as LaTeX or MathML with output format similar to natural handwriting. The software balances variability and style consistency in order to depict natural handwriting accurately.

In order to test our system we use samples created for the recent Competition on Recognition of Handwritten Mathematical Expressions (CROHME). Our software generates a sequence of strokes from a Symbol Layout Tree (SLT), a data structure used to represent the layout of mathematical expressions, by strategically sampling handwritten symbols to preserve a writing style, positioning these symbols relative to each other using information from the SLT, and applying distortions such as skew and rotation. Distortions are applied on each symbol, and on full expressions. The resulting strokes display a convincing representation of a handwritten expression.

The software itself is intended to be used for mathematical recognition systems built using machine learning techniques. These methods require large datasets for training, datasets that are currently not available. The web, however, contains many large collections of typeset mathematics, where formulas are encoded in LaTeX or MathML. Therefore, with our software and a relatively small training sample of handwritten mathematics, vast training and test datasets can be created from those collections.

Table of Contents

Chapter 1

Introduction

1.1 Motivations and Objectives

Recognizing handwritten mathematics is a difficult classification problem due to complex two-dimensional relationships and a large set of symbols to recognize. MathBrush is a system for recognizing handwritten mathematical expressions that uses information from symbol recognition with a grammar to produce valid mathematical expressions [1]. State of the art recognition systems use deep learning and neural networks for the recognition of online handwritten mathematical expressions. For example, Zhelezniakov et al. [2] use a bidirectional recurrent neural network with long short-term memory cell for symbol and structure recognition. They combine this with a context-free grammar to construct resulting expressions. Zhang et al. [3] use recurrent neural network with gated recurrent units and an attention mechanism to output a LaTeX expression. This method removes the dependency on a predefined grammar and would allow easier expansion and training. However, to build a robust recognizer that would recognize a wide variety of mathematical expressions of varying lengths and complexities, a very large number of diverse handwritten mathematical expressions are needed both for training and testing.

One dataset that exists comes from the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) [4], but this is limited both in scope and size. The 2019 training set consists of approximately 10000 expressions, and does not include examples with tabular structures like matrices and piecewise functions.

Enlarging an existing dataset by collecting more natural handwriting samples is the most straightforward option, but it has the downside of high cost. Generating a text document that looks like English handwriting can be easily done by creating a template for the set of English characters and common punctuation, and using this template as a font. This is not ideal for the two-dimensional nature of mathematics, and does not adequately reflect the format in which handwriting is collected. Moreover, a static font would have no variations

in the resulting expressions, unlike natural handwriting.

Data augmentation is commonly used in deep learning applications, and is effective when dealing with small or restrictive datasets. Training with data augmentation reduces the likelihood of a model overfitting the data, that is having a model that performs well on the training set but lacks the generalizability to match this performance on an unseen test set. Davila et al. [5] use an elastic distortion model to improve isolated symbol recognition by augmenting the CROHME dataset. Le et al. [6] utilize a decomposition and distortion strategy to create more, but smaller, expressions from existing expressions. Khuong et al. [7] devised a synthesizer that accepts LaTeX or MathML input and produces a natural looking handwritten mathematical expression with accurate ground truth data.

We are extending the synthesizer proposed by Khuong et al. to generate synthetic handwritten mathematical expressions, one which first converts the typeset expression into a symbol layout tree (SLT). The SLT is a structure that holds the spatial information of the target expression [8]. In a SLT, each node represents a symbol, and each directed edge represents the spatial relationship between the parent and child node.

By performing a depth first search on this tree and categorizing symbols by height-to-width ratio and relative baseline position, a mathematical expression layout is constructed. Normalized natural handwritten symbols are sampled from a dataset and inserted into the layout to produce a synthetic handwritten mathematical expression. Symbols are assumed to be clustered based on style, so duplicate symbols in an expression will only be sampled from one cluster.

In this work, we make several modifications to the synthesizer. We substitute an edge ordered recursive traversal in place of the depth first search layout creation in order to remove possible conflicts that may occur with long mathematical expressions. We query a table for symbol sizes to remove the need for symbol categorization. We extend our SLT to allow expressions with tabular structures to allow matrices and piecewise functions. We present a new method for style clustering used for whole expressions to enforce more style consistency. Finally, local and global distortion models are used to increase the variability of output expressions.

The objective of this project is to generate synthetic online handwritten mathematical expressions to assist in the training of a deep learning recognizer. A sufficient synthesizer must be able to create many versions of the same expression, as well as create new expressions from a markup language such as LaTeX or MathML with output format similar to how natural handwriting is collected. In addition, one needs to balance variability and style consistency in order to depict natural handwriting accurately.

The rest of this paper is structured as follows. Chapter 2 presents preliminary information, Chapter 3 provides details on previous work, Chapter 4 provides our contributions to the methods, and Chapter 5 proposes future work to further improve the synthesis of mathematical expressions.

Chapter 2

Datasets and the SLT

In this chapter we describe the CROHME isolated symbol dataset which we use for our natural handwritten symbols. We also describe the NTCIR dataset, a large set of typeset mathematical expressions in LaTeX form, which we use to test the robustness of our synthesizer. In addition, we discuss the details of the Symbol Layout Trees (SLT), a basic structure for representing layouts of mathematical expressions.

2.1 CROHME

The competition structure in [4] is composed of three tasks, online handwritten formula recognition, offline handwritten formula recognition, and formula detection in document pages. In the online handwritten formula recognition task, competitors are given a list of handwritten strokes gathered from a device such as a tablet and aim to produce the corresponding SLT. This task also has two subtasks, isolated symbol recognition, and formula parsing from given symbols. The isolated symbol recognition task contains a dataset of stroke data for individual symbols, as well as 'junk' symbols.

The input data for both formulas and isolated symbols are presented in InkML, an XML-based data format for representing ink entered with an electronic pen or stylus and a tablet. A handwritten trace is a list of two dimensional coordinates representing sampled points as a stroke is written. Each trace begins with the coordinates of a "pen down" event, and ends with the coordinates of a "pen up" event. Traces are grouped into symbols with appropriate labels, and annotations allow ground truth MathML to be included for access to structural information.

2.2 NTCIR MathIR

The NTCIR MathIR tasks participants to estimate the relevance of documents in a corpus to a query [9]. Queries are structured as a formula together with some keywords. One of the collections is a set of arXiv documents divided into paragraphs. Containing approximately 60 million math formulas from math, computer science, statistics, and physics, this dataset contains a wide variety of typeset mathematical expressions.

The example entry below consists of text with embedded mathematical expressions.

The free rotation is described now by the matrix operator $\mathbb{U}_r = \sigma_3 = e^{-i\frac{\pi}{2}(\sigma_3 - 1)}$ and the kick operator reads as $\mathbb{U}_k = e^{-iv\sigma_1}$. Simple manipulations with Pauli matrices lead to the following expression for the resonant Floquet operator

$$\mathbb{U}_{p,2}^{(res)} = e^{i\frac{\pi}{2}} \exp\left(-i\frac{\pi}{2}\mathbf{n}\cdot\boldsymbol{\sigma}\right) \quad (18)$$

where the unit vector $\mathbf{n} = (0, \sin v, \cos v)$. Up to the trivial phase factor, this is a spin-flip operator which belongs to the unitary unimodular group $SU(2)$. Therefore, corresponding evolution in the continuous time t fully reduces to the spin rotation.

In our case the data is useful since we can extract the typeset LaTeX formulas and can establish the range of functionality of our synthesizer and examine failure points of the system.

2.3 Symbol Layout Tree

A SLT [8] is a representation of a mathematical expression that encodes the spatial relationships between symbols. Each node of a SLT represents a symbol, while labelled edges represent the positional relationship between a parent and child node. As this is a tree structure, every node, other than the root node, has exactly one parent.

For a mathematical expression, we can take a markup language input that encodes the visual presentation of a formula and extract the SLT information from the relevant markup syntax, taking the first occurring symbol as the root of the tree.

In the case of the CROHME formula dataset, six edge types are needed. The naming conventions used for the edge types in this work are: next, above, below, over, under, and within. The next edge type represents a horizontal relationship where the child expression is placed to the right of the parent symbol. Above characterizes superscripts, that is, a scaling down of the symbol size, and translating the child expression so it resides above the vertical midpoint, and to the right of the parent. Similarly, below characterizes subscripts,

by translating the child expression to reside at or below the baseline of an expression to the right of the parent. Over represents a child expression with horizontal positioning centred relative to the parent, but vertically translated so the child's minimum vertical value is greater than the parent's maximum. Under retains the same horizontal positioning as over, but with vertical translation so the child's maximum vertical value is less than the parent's minimum.

The within edge is a special case used for radical expressions, depicting that the child expression is the radicand. This requires that the parent node be scaled and positioned, unlike the other edge types. Firstly, the radical part must be vertically scaled to have height greater than that of the radicand. Then the viniculum (or overline) must be scaled horizontally to be at least the same width as the radicand. Finally, the child expression must be placed to the right of the radical part so the maximum vertical value of the child is less than the height of the overline.

These 6 edges can be used to describe a wide variety of mathematical expressions, but would not be good enough to handle all mathematical expression. Examples of the SLT are provided in Figures 2.1 and 2.2. We require extensions that are described in chapter 4 to extend the scope of our SLT beyond the CROHME dataset. Since tools exist to convert markup language input to a SLT structure, the remainder of this work will assume the starting point for data synthesis will be the SLT of a mathematical expression.

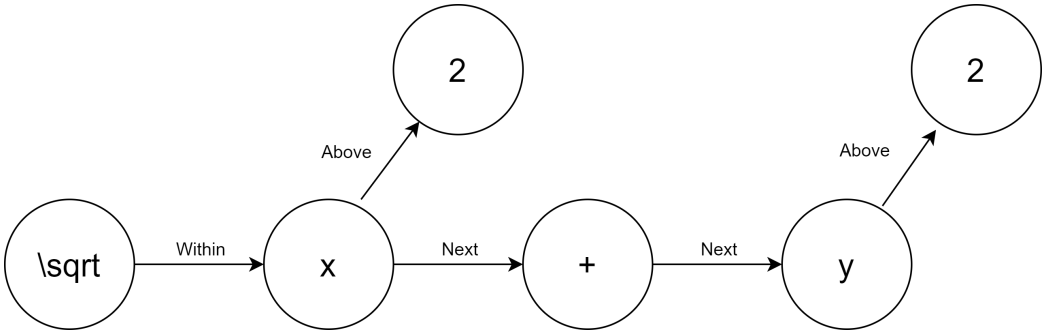


Figure 2.1 SLT for the expression $\sqrt{x^2 + y^2}$

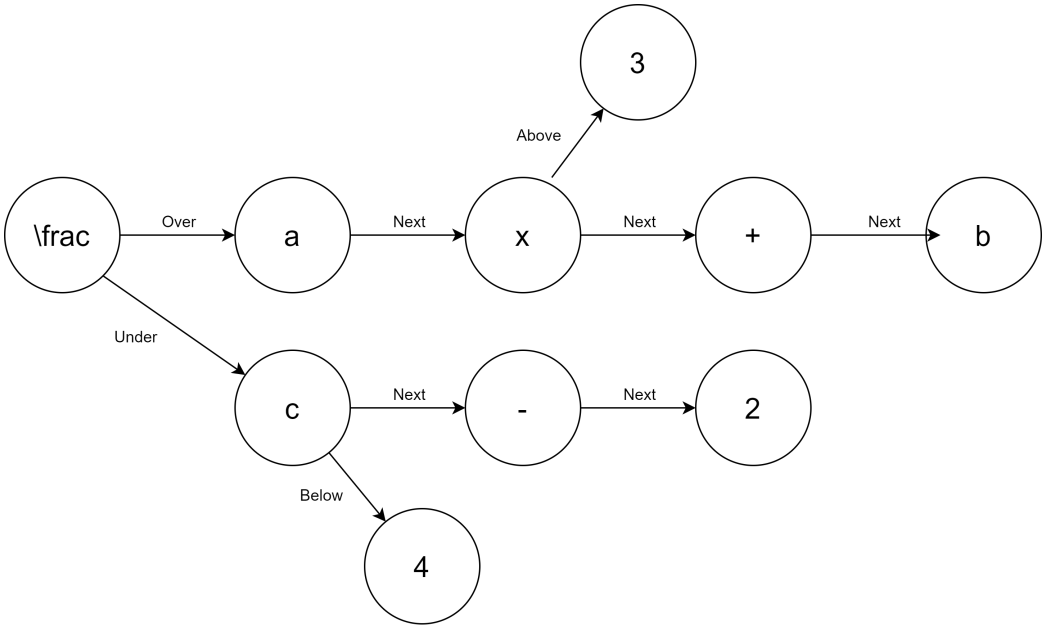


Figure 2.2 SLT for the expression $\frac{ax^3+b}{c_4-2}$

Chapter 3

Previous Work

In this chapter we explore previous work in data augmentation and synthesis. We discuss an expression decomposition strategy to augment the CROHME formula dataset by isolating sub-expressions as a preliminary step towards synthesis. We look at local and global distortions used in the training of state of the art recognition systems for the purpose of use in our synthesizer. We also examine warp grid distortion used for English text images and assess the usage for mathematical expressions. Finally we survey a method to produce a synthetic handwritten expression from markup languages and examine issues that can arise.

3.1 Mathematical Expression Decomposition

Le et al. [6] use a decomposition and distortion strategy to extend the CROHME formula dataset for improving an end-to-end deep learning recognition system. Since the CROHME dataset groups traces into symbols, a handwritten mathematical expression can be split into multiple sub-expressions or individual symbols by decomposing mathematical expression grammar rules. By constructing a SLT from a target expression, they uses four rules to decompose the expression into valid mathematical sub-expressions:

1. If an expression contains any above or below edges, make a baseline sub-expression by removing those edges.

2. Create a sub-expression starting from the child of each above, below, over, under and within edges in the SLT.

3. Generate two sub-expressions for the left and right parts of every binary operator in the baseline expression generated by rule 1. We can make the left sub-expressions from the SLT by removing next edges where the child symbol is a binary operator, and the right sub-expression by starting with the child node whose parent is a binary

operator. Note that sub-expressions are not created if the operator resides inside brackets to guarantee resulting expressions are valid.

4. Discard singleton expressions.

So given the expression $ax_1^{2n+1} + by_1^2 - cx_2 + dy_2$, the rules would yield sub-expressions

1. $ax + by - cx + dy$, the baseline sub-expression.
2. $2n + 1$, 2, 1, from the superscripts and subscripts
3. ax, $by - cx + dy$, $ax + by$, $cx + dy$, $ax + by - cx$, dy, by splitting the binary operators from rule 1.
4. Remove singleton expressions 1 and 2 from rule 2.

Although this decomposition technique can provide many more samples to augment a training dataset, we note that decomposing always reduces the total token count of the sub-expression. This means that the scope of the expressions in the augmented training set will gain little expression variety.

This method had been shown in [6] to improve the recognition rate of a deep learning system. There is then a large variety of expressions that can be created from a synthesizer which is built from a markup input and which returns a synthetic expression.

3.2 Local and Global Distortions

Le et al. also extend a distortion strategy used by Chen et al. [10]. This distortion model uses both local (symbol) and global (expression) distortions to increase the variability of the input data. Note that some of these transformations are commonly used for image augmentations due to the ease of implementation and "safety" of the transformations [11]. A safe transformation will retain the label of the initial data, and some transformations only remain safe if certain parameters of the mapping are enforced. Rotations of images can augment a dataset of objects without changing the label of an image. However this should be avoided if using numbers, since rotating a 6 can turn it into a 9. The distortions used at the local level are shear, shrink, perspective, shrink followed by rotation, and perspective followed by rotation. At the global level, shear and rotation are used. Figure 3.1 provides a visual representation of these transformations.

Where (x, y) is a single point in a trace, and (x', y') is the transformed coordinate, they present the equations of the transformations with parameter θ.

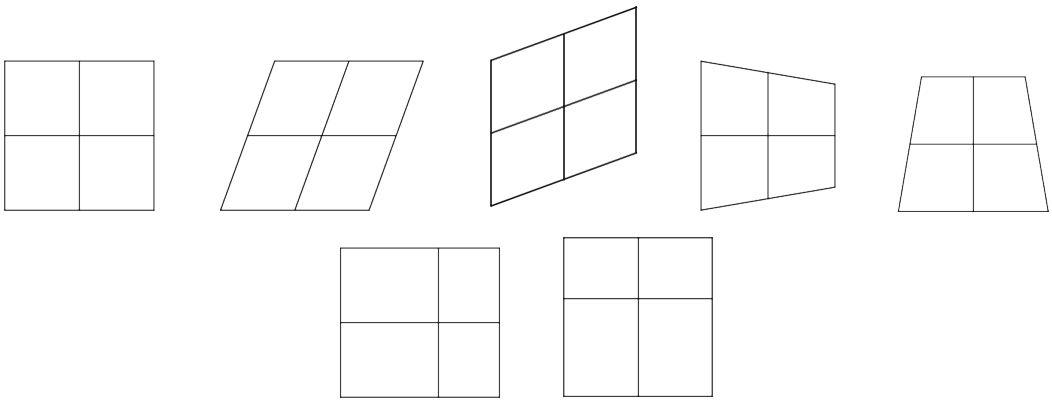


Figure 3.1 Square with side length 100 under a horizontal shear, vertical shear, horizontal shrink, vertical shrink, horizontal perspective, and vertical perspective transformations

Transformation	Horizontal	Vertical
Shear	$x' = x + y\tan\theta$ $y' = y$	$x' = x$ $y' = x\tan\theta + y$
Shrink	$x' = x$ $y' = y\left(\sin\left(\frac{\pi}{2}-\theta\right) - \left(\frac{y\sin(\theta)}{100}\right)\right)$	$x' = y\left(\sin\left(\frac{\pi}{2}-\theta\right) - \left(\frac{y\sin(\theta)}{100}\right)\right)$ $y' = y$
Perspective	$x' = \frac{2}{3}\left(x + 50\cos\left(4\theta\frac{x-50}{100}\right)\right)$ $y' = \frac{2}{3}y\left(\sin\left(\frac{\pi}{2}-\theta\right) - \left(\frac{y\sin(\theta)}{100}\right)\right)$	$x' = \frac{2}{3}x\left(\sin\left(\frac{\pi}{2}-\theta\right) - \left(\frac{x\sin(\theta)}{100}\right)\right)$ $y' = \frac{2}{3}\left(y + 50\cos\left(4\theta\frac{y-50}{100}\right)\right)$

The rotation model is:

$$x' = x\cos\theta - y\sin\theta, \quad y' = x\sin\theta + y\cos\theta.$$

To generate an expression, five variables are randomized for the distortion model, $id, \theta_0, \theta_1, \theta_2, k$. The variable $id \in 1, ..., 5$ represents one of the local distortions mentioned earlier. The three variables $\theta_0, \theta_1, \theta_2 \in [-\pi/18, \pi/18]$ are input parameters for individual distortions. θ_0 is the input parameter for shear, shrink, and perspective. θ_1 and θ_2 are the input parameters for local and global rotations respectively, and $k \in [0.7, 1.3]$ is the global scaling factor. All symbols in an expression are then distorted by local model, after which the global distortions are applied.

The hybrid approach from [6] first uses the decomposition method to generate sub-expressions, and then applies a distortion model. CROHME participants USTC-NELSLIP and iFLYTEK Research [3], and CASIA/NLPR PAL Group [12], use this hybrid strategy to augment the given dataset, with the former scoring the highest recognition rate among competitors.

This distortion approach has the downside of applying the same local distortion to each object. Natural handwriting will not have characters which are deformed uniformly.

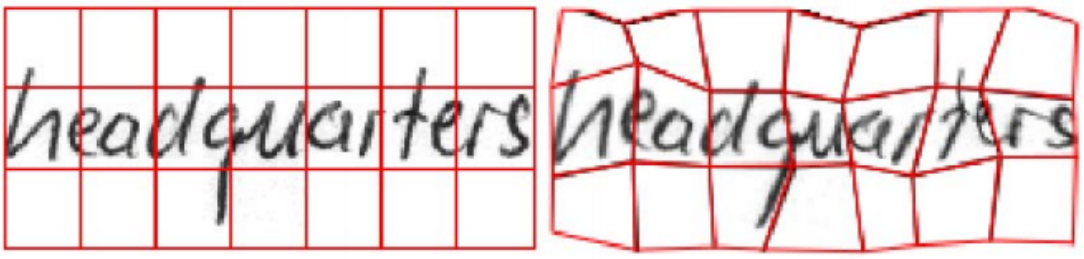


Figure 3.2 Word image with uniform grid, and perturbed grid with distorted image (Wigington 2017)

3.3 Random Warp Grid Distortion

In [13], the authors observe that variations in natural handwriting are not uniform affine transformations across a word. Each character has a slightly different scale and slant. This leads them to develop the random warp grid distortion (RWGD). First, control points are placed on a regular grid aligned to the baseline. Each control point is then perturbed by sampling from a normal distribution. Finally, the image is warped in relation to the perturbed control points as shown in Figure 3.2.

For the purpose of handwritten words, the control points are specifically placed so the middle row of the grid approximates the heights of lower case letters, to apply the warp over entire characters. In our case, where the traces of a specific object can be isolated, we can perturb the values of a reference box and warp the pen points accordingly. Unfortunately, the method of how to warp the inside of a grid section is left undisclosed.

3.4 Generating an expression from a SLT

Khuong et al. [7] propose a method of building a synthetic handwritten mathematical expression from a SLT. First, the SLT is used to construct an expression layout. The CROHME isolated symbol dataset is then used to sample handwritten symbols which are positioned according to the layout.

A layout of a synthetic expression is constructed using the position and size of each object in the SLT. Both of these quantities can be defined by the bounding box of the character. All unique symbols from the sampling dataset are grouped based on relative position of the symbol with the baseline and mean line, as well as the width to height ratio.

Relative position has four groupings. Normal symbols occur between the baseline and mean line. Small symbols also occur between the baseline and mean line, but are very small in size. Ascendant symbols extend above the mean line. Descendant symbols extend below the baseline.

Edge Type	s_ratio	r_point	s_point	dx	dy
Next	1.0	Bottom-Right	Bottom-Left	cs/5	0
Below	0.4	Bottom-Right	Top-Left	cs/10	cs/3
Above	0.4	Top-Right	Bottom-Left	cs/10	cs/3
Over	0.3-0.7	Top-Left	Bottom-Left	0	cs/5
Under	0.3-0.7	Bottom-Left	Top-Left	0	cs/5

Table 3.1 Size and position of the Child Symbol Based on the Parent. Adapted from [7]

Width and height ratios are divided into six classes: approximately equal, height greater than width, height much greater than width, width greater than height, width much greater than height, and variable. Note that variable is only used for radicals. On the CROHME 2016 dataset, these two groupings form 12 categories using the 101 symbols.

The object in the root node of the SLT is initialized to have a default height of 100 pixels and bounding box bottom-left position at $(0, 0)$. The sizes and positions of the rest of the entities in the SLT are determined based on the relation to their parent. A depth first traversal is employed to size and position the symbol in the child node relative to the parent.

Each edge type is given five values to assist in the positioning of the layout as described in Table 3.1. The size ratio (s_ratio) between the size of the child symbol (cs) and parent symbol, set point (s_point), reference point (r_point), horizontal distance between s_point and r_point (dx), and vertical distance between s_point and r_point (dy) are used. The set and reference points are the corners of the child and parent bounding boxes respectively.

The over and under relations are aligned to the parent after positioning all the objects in the sub-expression of the child node. This allows the alignment of the child with the parent for a more natural looking expression. The fraction bar is a special case, since the width will be scaled relative to the sizes of the numerator and denominator.

During the layout creation process, the parameters listed in Table 3.1 are perturbed to increase the layout variety. The layout process terminates when every node of the SLT has been reached by the depth first search.

Handwritten symbols are then sampled for placement in the layout. Note that the authors in [7] assume that samples are clustered based on style for each character, so repetitions will be sampled from the same style class. Samples are normalized before positioning into the layout. This normalization occurs because of differences in bounding box sizes between the layout and the sampled symbols and preserves the original sample's height to width ratio.

Radicals using the within relation are considered a special case since height and width are dependent on the inner expression. Khuong et al. divide this handwritten symbol into the "overline" and the "radical" before normalization. Multi-stroke samples are first merged to form a single stroke. Valley point detection is then used to find the lowest point of the vertical segment of the radical. From that point, each line segment connecting consecutive

points is examined. The first line segment that forms a 20° angle or less with the horizontal axis determines the beginning of the overline.

Khuong et al. performed an experiment where volunteers rated how natural both synthetic and human written expressions looked. This method was shown to generate natural looking synthetic samples, however a significant portion of the synthetic patterns had unnatural points that were commented on by evaluators. Five general problems are noted: Inconsistency in writing style of symbols, too small or large symbol sizes, ambiguity in the relationship of objects, strange writing style, and unnatural distance between objects. The authors believe that these problems are caused by the random parameters, symbol normalization, and the choice of samples.

Other concerns from this method are: (1) Under the principle that a layout is created only using information from the parent and child symbols, it is possible that there could be significant overlap between entities, which would not be representative of natural handwriting. Consider the expression $2^{3x^2-4x-1} + 3$. With the current layout construction rules, it is likely that the long superscript would extend over the baseline expression causing a very unnatural look; (2) Forcing a radical to be a single stroke may not be consistent with most authors, and would skew the dataset; (3) The categorization of all objects in the dataset may not be easily extendable if future datasets contain more variety; (4) Style clustering for individual symbols is assumed, however no method is proposed.

Chapter 4

Handwritten Math Synthesizer

In this chapter we discuss the design and implementation of our synthesizer. In [7], a layout of the mathematical expression is constructed first, followed by the insertion of samples. We will sample and scale the objects first, and recursively construct the handwritten mathematical expression.

We use the CROHME isolated symbol dataset to sample and scale symbols used in an input expression. We discuss our normalization of the samples using bounding box information from a Unicode font. We also discuss normalization performed on the SLT nodes to remove unwanted typographical information and allow custom composite objects that occur in our sample set but not in our font. We describe the extensions made to the SLT to allow more complex mathematical structures.

We provide information about the edge ordered recursive traversal of the SLT, with details on how each edge merges the child with the parent node. We detail the implementation of distortions used on the character level to increase variability. Finally we establish two criteria for symbol selection to ensure style consistency. We propose two methods of selection based on metadata.

4.1 Formatting Symbol Samples

We avoid categorizing a specific set of objects by querying a Unicode font for spatial symbol information. Here we use the Asana-Math font [14], which by default contains character glyphs in 1000 point font. We extract five values for each object defining the bounding box with the minimum and maximum values along the $x-$ and $y-$axes denoted $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$, and the width of empty space to be inserted to the right of a symbol, called the right side bearing (rsb). This information removes the need to group based on height to width ratio, and since the $y-$axis values are relative to the baseline, also means that the

relative positioning classification is unnecessary. In addition, these values also yield easy access to an object's height, width, and midpoints which will allow for easier positioning.

For a given SLT, each node is assigned a list of traces from the isolated symbol dataset. From the font bounding box information, the sample is scaled to fit inside the bounding box. If the height of the sample is greater than the width, then the sample is scaled to the height of the bounding box, retaining the height to width ratio of the original sample. Otherwise, the opposite is performed. After scaling, the new bounding box is calculated.

4.2 Formatting Input SLT

Usage of the font file however does not give bounding boxes for all mathematical entities that can occur in a SLT. Trigonometric functions, logarithms, and limits are treated as individual objects. For these expressions that also exist in the isolated symbol database, such as sin and cos, we create custom bounding boxes to extend the reach of our font. For more specialized operators that do not have samples such as inverse trigonometric functions like arctan, the SLT returns the text "arctan" and we can construct the function name using individual letters when possible.

The breadth of symbols used in the NTCIR dataset poses conflicts with the isolated symbol dataset. When creating a SLT, character variants can be returned depending on the input string. Italicized, bold, and other font specific entities are represented by their own unique character codes. Unicode provides two types of character equivalences [15]. Canonical equivalence occurs when two characters or sequences of characters represent the same abstract character, and have the same visual appearance. Compatibility equivalence occurs when two characters or sequences of characters represent the same abstract character, but have a different visual appearance. Figure 4.1 shows the letter "A" and multiple variations with unique character codes. Unicode Normalization Forms are used to determine equivalence between Unicode strings. Since the usage of these stylistic properties is limited for human input, all symbols are compatibility normalized before any sampling occurs.

Figure 4.1 Unicode characters corresponding to normal, bold, italic, sans serif, and script. These characters all compare equivalent under compatibility normalization.

4.3 Extended SLT

Accurate description of a wider variety of expressions requires more spacial relationship information than the six SLT edges types used in [7]. For this purpose we use a SLT that gives a relationship for the degree of a radical. This above-like edges depicts superscripts occurring to the left of the parent symbol specifically in the case where the degree is not two.

To support matrices and piecewise functions, the requirement that a node of the SLT must be a symbol is adjusted. Davila et al. [16] extend a node to include a tabular structure holding the number of rows and columns, and the type of bounding brackets. Note that no brackets or a one sided left-bracket are also permitted.

A matrix node must have a within edge connected to the upper left corner element of the matrix. A next-element edge is introduced to connect the elements of the matrix. For an $m \times n$ matrix, the parent element at position (r, c) will have next-element child at $(r, c+1)$ if $c + 1 < n$, and at $(r + 1, 0)$ otherwise.

4.4 Edge Ordered Recursion

In our recursive implementation, the traces from each child node are absorbed into the parent node in an order based on edge type. The edges are traversed and merged into the parent object with a priority. First, tabular objects with the matrix structure are combined into a single node. Then radicals are processed with the within edge, followed by the degree edge. The degree is always combined with the radical expression after the scaling of the radical, otherwise, the scaling of the degree would not preserve the height to width ratio. Above and below are next to be processed and require that both be merged with the parent before the right side bearing of the node is adjusted to prevent an unnecessary shift in the subscript. Over and under are then positioned, with a special case for the fraction bar. The edge type next is the last to be merged with the parent. This ensures that the parent sub-expression is completely rendered before positioning an adjacent expression.

4.5 SLT Edges

First we look at the different edge types and discuss how a parent and child expression connected with this edge type will be merged into a single object. For all edge types, the points of the child node (x, y) are scaled and positioned to (x', y') before extending the parent list of traces with the traces of the child. The bounding box information is recalculated whenever a child is merged into its parent node. We also perturb each translation and

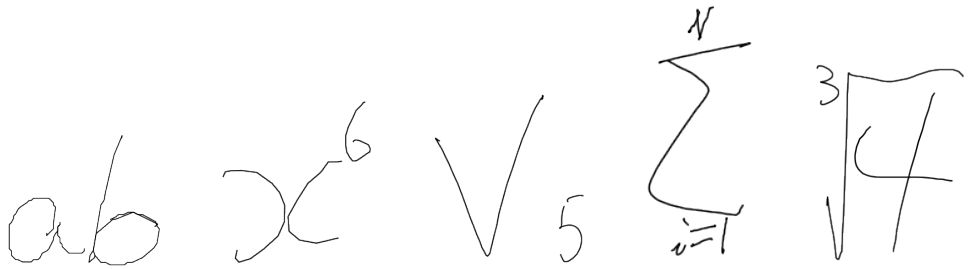


Figure 4.2 Synthetic generated samples demonstrating the edges: next, above, below, over and under, within and radical degree

scaling factor by sampling a small value denoted here as ε_i from a normal distribution to increase variability.

Next

The edge type Next denotes a horizontal adjacent relationship between the parent and the child.

$$x' = x + \text{parent}[x_{\max}] + \text{parent}[\text{rsb}] + \varepsilon_x$$
$$y' = y + \varepsilon_y$$

In other words, the child is shifted horizontally so all of the traces are to the right of the parent, with no vertical shift.

Above and Below

The edge type Above denotes a superscript relationship between the parent and the child.

$$x' = Cx + \text{parent}[x_{\max}] + \frac{1}{2}\text{parent}[\text{rsb}] + \varepsilon_x$$
$$y' = Cy + \text{parent}[y_{\max}] + \varepsilon_y$$

The edge type Below denotes a subscript relationship between the parent and the child.

$$x' = Cx + \text{parent}[x_{\max}] + \frac{1}{2}\text{parent}[\text{rsb}] + \varepsilon_x$$
$$y' = Cy + \varepsilon_y$$

These edges shift a scaled version of the child sub-expression to be to the right of the parent expression but closer than that of the Next edge. Vertically, the Above child is shifted

to the top of the parent, and the Below child continues to reside on the baseline. For both these edge types, we choose scaling factor $C = 0.3 + \varepsilon_C$. In addition, after both of these edges are merged with the parent, we adjust the parent's right side bearing to 90, a default spacing value given by our font.

Over and Under

In the Over edge case the symbol is first scaled, and then horizontally centered, and vertically positioned higher than the parent symbol.

$$x_{\text{new}} = Cx + \frac{1}{2}(\text{parent}[\text{width}] - \text{child}[\text{width}]) + \text{parent}[x_{\min}] + \varepsilon_x$$
$$y_{\text{new}} = Cy + \text{parent}[y_{\max}] - \text{child}[y_{\min}] + D + \varepsilon_y$$

Similarly, the edge type Under is scaled, and then horizontally centered, and vertically positioned lower than the parent.

$$x' = Cx + \frac{1}{2}(\text{parent}[\text{width}] - \text{child}[\text{width}]) + \text{parent}[x_{\min}] + \varepsilon_x$$
$$y' = Cy + \text{parent}[y_{\min}] - \text{child}[y_{\max}] - D + \varepsilon_y$$

For both these edge types, we choose $C = 0.33 + \varepsilon_C$, and $D = 50$.

As in [7], the fraction bar is a special case of applying both an Over and Under relationship simultaneously. Scaling of the parent fraction bar width must be applied first, followed by the translations but not the scaling of the Over and Under children as above. Specifically we scale the width of the parent to be the greater of the two over and under children's widths.

Within

The Within edge is a special case that applies only to radicals. As in [7], the radical must be scaled both vertically and horizontally to fit the expression within, however, we explore an alternative solution to this problem. Here we look at finding the maximal empty rectangle inside the bounding box of the radical. Given the height, width, and lower left corner of this rectangle, we can vertically scale the entire symbol to match the required height of this child expression, and use the left side of the rectangle to establish the start of the overline, and horizontally scale this portion of the symbol to match the width of the inner expression.

We find an approximate maximal empty rectangle as follows. First, we set up an $n \times n$ grid encapsulating the radical. For each grid section, we mark the section with a 1 if there

0	0	1	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	1	1
1	0	0	0	0	0
0	0	0	0	0	0

1	1	0	1	1	1
0	2	1	2	2	2
1	3	2	3	3	3
2	4	3	4	0	0
0	5	4	5	1	1
1	6	5	6	2	2

1	2	0	1	2	3
0	1	2	3	4	5
1	2	3	4	5	6
1	2	3	4	0	0
0	1	2	3	4	5
1	2	3	4	5	6

Table 4.1 Example Binary Array with corresponding Height and Width Arrays

exists a point from the symbol which resides inside that grid section. This yields a 2-D binary array, with our goal shifting to find the largest rectangular sub-array of zeros.

We solve this problem by passing through each element of the array twice. On the first pass, establish a height array, H. Traversing each column, we note the number of consecutive 0s up to the current element. On the second pass, we similarly traverse over the rows, noting the number of consecutive 0s up to the current element, creating a width array W. Table 4.1 gives an example of the results of these traversals. Now for each non-zero entry occurring in both the height and the width array, we can compare the area of all rectangles with bottom right corner by looking at all possible heights, and taking the product with the smallest width in that height range. That is, for an element in row r and column c:

$area_{max}, h_{max}, w_{max} \leftarrow 0, 0, 0$
for $h \in 1, \ldots, H[r, c]$ **do**
  $w \leftarrow \min(W[r - h + 1, c], \ldots, W[r, c])$
  $area \leftarrow w \times h$
  if $area > area_{max}$ **then**
    $area_{max}, h_{max}, w_{max} \leftarrow area, h, w$
  end if
end for
return $area_{max}, h_{max}, w_{max}$

From these results we can obtain the global maximum area rectangle, and the start of the overline part of the radical. This also removes the need to make the radical a single stroke, retaining the variability of the input symbols.

Radical Degree

Another small addition to the SLT is the inclusion of an edge to position the degree of a radical. Similar to the Above relation, we transform the symbol as follows:

$$x' = Cx + \text{parent}[x_{\min}] + \frac{1}{2}\text{child}[x_{\max}] + \varepsilon_x$$
$$y' = Cy + \frac{3}{4}\text{parent}[y_{\max}] + \varepsilon_y.$$

In the horizontal direction, we try to align the right most point of the radical degree to the radical's leftmost point. Instead of aligning the vertical component with the top of parent object, the symbol is lowered to better mimic a human written expression.

4.6 Matrices and Piecewise Functions

Each element of the matrix has all its standard edges processed before being positioned into a two dimensional array with the same shape as the target matrix. After this array is filled, the $y_{\max}$ and $y_{\min}$ values of all elements of a row are set to be the $y_{\max}$ and $y_{\min}$ of the entire row. Similarly, the $x_{\max}$ and $x_{\min}$ values of all elements of a column are set to be the columns corresponding maximum and minimum.

The rows are then assembled by connecting the element from column c to column $c + 1$ using a next edge. The rows are stacked on top of each other using the under edge with scaling factor $C = 1$. Finally, the brackets are scaled to the height of the result, and the left and right extents are adjusted accordingly.

4.7 Distortions

In [13], it is noted that word images augmented with shear or rotation do not represent the natural variations in handwriting. Rather, small differences in slant and scale vary between characters in each word, as opposed to uniform slants caused by affine transformations. While their method focuses on distorting by perturbing a set of grid points in a word image, we can apply the same concept by perturbing the corners of a reference box, and warping the points of a symbol using a projective transformation. Such a mapping must be non-linear.

For the two dimensional case, given a set of n source points, (x_i, y_i) and corresponding destination points (x'_i, y'_i) , we desire to find constants $a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2 \in \mathbb{R}$

such that

$$x'_i = \frac{a_0 x_i + a_1 y_i + a_2}{c_0 x_i + c_1 y_i + 1}$$
$$y'_i = \frac{b_0 x_i + b_1 y_i + b_2}{c_0 x_i + c_1 y_i + 1}$$

for all $i \in 1, \ldots, n$. Rearranging, the unknown coefficients form a linear system with $2n$ equations:

$$0 = a_0 x_i + a_1 y_i + a_2 - c_0 x_i x'_i - c_1 y_i x'_i - x'_i$$
$$0 = b_0 x_i + b_1 y_i + b_2 - c_0 x_i y'_i - c_1 y_i y'_i - y'_i$$

or in matrix form

$$\mathbf{M}\vec{z} = 0$$

where $\mathbf{M} \in \mathbb{R}^{2n \times 9}$ has rows

$$M_{2i-1} = \begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x'_i & -x_i x'_i & -x'_i \end{bmatrix}$$
$$M_{2i} = \begin{bmatrix} 0 & 0 & 0 & x_i & y_i & 1 & -x_i y'_i & -y_i y'_i & -y'_i \end{bmatrix}$$

for all $i \in 1, \ldots, n$, and

$$\vec{z} = \begin{bmatrix} a_0 & a_1 & a_2 & b_0 & b_1 & b_2 & c_0 & c_1 & c_2 \end{bmatrix}^\top.$$

Here we solve the total least squares problem. That is, we aim to find $\vec{z}$ that minimizes $\|\mathbf{M}\vec{z}\|_2$ such that $\|\vec{z}\| = 1$, and scale the solution to enforce $c_2 = 1$. The solution to this problem is the right-singular vector of $\mathbf{M}$ corresponding to the smallest singular value. We can factor $\mathbf{M}$ with singular value decomposition:

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

so for $m \leq \min(2n, 9)$, we obtain orthogonal matrices $\mathbf{U} \in \mathbb{R}^{2n \times m}$, $\mathbf{V} \in \mathbb{R}^{9 \times m}$ with columns corresponding to the left and right singular vectors of $\mathbf{M}$ respectively, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times m}$ is a diagonal matrix, with entries consisting of the non-zero singular values of $\mathbf{M}$. Note here that we can choose the decomposition such that $\mathbf{\Sigma}_{11} \geq \mathbf{\Sigma}_{22} \geq \ldots \geq \mathbf{\Sigma}_{mm}$. In this case, right-singular vector corresponding the smallest singular value of $\mathbf{M}$ is the m-th column of $\mathbf{V}$.

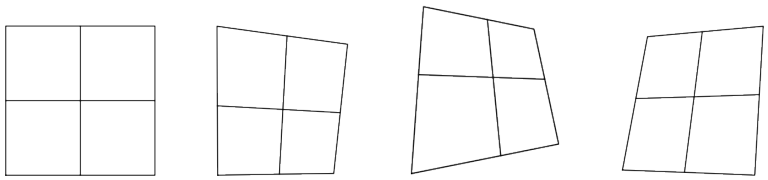


Figure 4.3 Warp grid distortion reference box, with random perturbations

In the general case using this method, it is possible for the total least squares solution to have a c_2 value close to zero. This corresponds with trying to map a plane to a line and can be avoided by restricting the amount we perturb the reference box. The reference box is a square with lower left corner at $(0, 0)$, with side length equal to that of our font size (1000). Each coordinate of every corner is perturbed by random noise from a $\mathcal{N}(0, 1000 \times d)$, where $d = 0.1$ in our default implementation. Figure 4.3 shows a variety of distortions that can be applied using this method.

We employ a distortion model similar to [6]. The local distortions used are vertical and horizontal shear, rotation, scaling, and warp grid distortion. A single distortion is randomly chosen for each individual symbol as it is being added into the tree structure.

4.8 Clustering

Khuong et al. assume style clustering of individual symbols, so when a symbol is repeated in an expression, a sample is pulled from the style cluster and not at random for style consistency. This however does not enforce any style consistency across the whole expression. We explore methods to cluster the data with two main preferences to enforce style consistency over an entire expression. Given a starting symbol, we prefer to sample symbols written by the same author to preserve style. If an author has not written a particular entity, we prefer a sample taken on the same input device, or by the same collecting institution. Clearly, style consistency will improve under the first preference. We can expect a slight improvement in style consistency with the second preference since each device may handle an input slightly differently, and post-processing done by the collectors may be performed using different methods.

4.8.1 Naive Clustering

The CROHME dataset comprises handwritten samples from a variety of different collections. These collections are appended to each other to create one large dataset. This means that objects written by the same author as a part of a collection will be close to each other by index. We can naively sample by selecting the symbol that is closest by index to one

previously sampled. While this works well for symbols which fall in the middle of a block of samples, if the sample is close to the beginning or end of a section, the next closest symbol may well be from a different author or input device.

4.8.2 Metadata Clustering

We note that the provided isolated symbol database also contains metadata strings. These strings are unique to the sample and typically identify the collecting institution, the author, and an enumeration of the number of symbols collected. Table 4.2 provides examples of metadata strings taken from CROHME. Using a string comparison, we can increase the likelihood that a sample will be taken from the same author or input device. Note that these metadata strings are of variable lengths, so a naive string comparison of characters in the same position will not work effectively. The Levenshtein distance [17] can compare strings of different lengths and cluster our data with respect to our preferences.

Given strings a and b, their Levenshtein distance $\text{lev}(a, b)$ is defined as

$$\text{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

Intuitively, the Levenshtein distance is the minimum number of single-character edits, whether they be insertions, deletions, or substitutions, required to change one string into the other. Sampling child symbols with metadata that has a lower Levenshtein distance to the parent symbol's metadata will increase style consistency by ensuring that symbols are handwritten by the same author, or collected by the same input device, whenever that symbol is available. In the event of a duplicate symbol, the identical sample may be returned, but the instances will not look identical due to the implemented distortion models.

2011_IVC_depart_F015_E007_20293
2011_IVC_depart_F029_E045_30370
MfrDB1440_69070
85_Frank_7743
85_Frank_7753

Table 4.2 Examples of metadata strings in the CROHME 2019 dataset

4.9 Workflow and Testing

Our system accepts input as a file or string of LaTeX or MathML. LaTeX is first converted to MathML using the LaTeXML library [18]. The SLT edges are extracted from MathML with an in-house tool, and a Python class to hold the tree structure is assembled. This class selects and distorts samples pulled from the CROHME dataset as lists of NumPy arrays [19], and performs a recursive call to absorb the traces from the children into their parent nodes by appending the trace list of the child onto that of the parent. After all edges have been processed, the trace list of the root node contains the synthesized expression. We output this list either as is, or convert the result into InkML. See Appendix A for examples.

We stress test the system on the wide variety of LaTeX expressions from the NTCIR dataset. Initial failed tests led to the Unicode character normalization mentioned previously. Additionally, our SLT returned invisible Unicode characters. Invisible multiplication and function applications are syntactic elements used in Content MathML which have no influence in our case. A bounding box with zero height and width, and an empty list of traces are used when constructing an expression with these problematic elements.

Out of 10,000 LaTeX expressions, our system succeeds in generating 85% of the expressions, however roughly half of the successes substitute bounding boxes for a missing handwritten sample. This result suggests that the dataset from CROHME does not contain enough variety to adequately represent the breadth of mathematics in academic use. Failures occur when our tool for MathML to SLT conversion does not provide a valid output, neither the font nor our custom tables have bounding box information for a symbol, or when an empty character exists in the expression.

Chapter 5

Conclusions and Future Work

In this paper, a model for the generation of synthetic online handwritten mathematical expression from a SLT was examined and enhanced. The enhancements include replacing the depth-first search layout construction with an edge ordered recursive building method for the synthetic expression. The ability to obtain symbol bounding box information from a font instead of requiring symbol categorization improves the models ability to adjust to new input datasets without the need to manually categorize new symbols. Local distortions were used to increase the variability of the model output. Metadata based sampling improves the style consistency by ensuring samples are taken from the same author or input device when possible. Lastly, we extend the previous method by adding support for tabular expressions such as matrices and piecewise functions.

Although these improvements widen the scope of the generative model, much more can be done in the way of data synthesis for handwritten mathematical expression. Firstly, we still rely on directly sampling from an existing dataset. A method which generates individual symbols given some style constraint would be an asset. As previously stated, only 101 distinct symbols are available in the CROHME dataset. A synthesizer that creates handwritten symbols that are not included in the training dataset would provide a great tool to enhance the scope of possible expressions. Additional distortion models can be added to this system to further increase variability.

Recurrent neural networks using long short-term memory cells [20] can generate realistic English handwriting in a variety of styles. Combining this method with the SLT may be possible by integrating a Tree-Structured approach [21]. Alternatively, generative adversarial networks have been used to synthesize handwritten text images [22] that could be applied to either a full mathematical expression or individual symbols, however, this would result in static image, as opposed to a sequence of points.

Statistical stroke-based models have also been used to generate symbols in a variety of languages [23] using limited training samples. This method uses individual stroke

models combined with a character level interstroke model. Strokes are synthesized based on the individual stroke model, then positioned and sized based on the interstroke model for characters. Classification of the strokes is performed by analyzing the vectors between points that are visible to each other. This classification method would be useful to cluster our target dataset into stylistic groups.

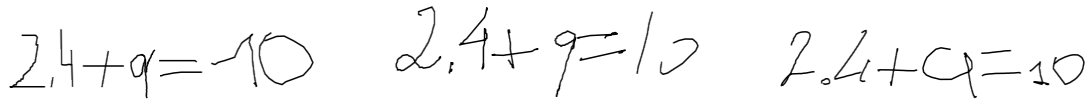
While using the NTCIR database for a wide range of mathematical expressions serves to act as a robust stress test for our generation process, the end goal of training a recognizer for mathematics might be better suited to a more restricted dataset for the expected type of input expressions. The development of a grammar to automatically generate LaTeX expression would be an asset in this case. Alavi et al. [24] propose a rule-iterated context-free grammar for the production of specific mathematical expressions. Such a grammar could be used in tandem with our synthesizer to address a specific recognition use case, or to target problem areas of existing recognition technology.

Appendix A

Synthetically Generated Samples

```
\int_{x = 3}^{6} \cos{\left[\pi \theta \right]} d \theta
```

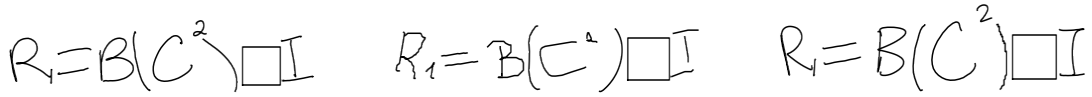
```
2.4 + q = 10
```



```
\sqrt{b^2 - 4ac}
```

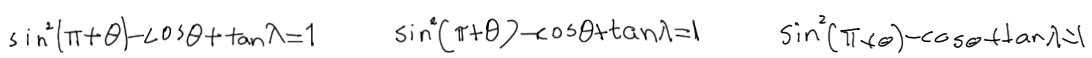
```
\begin{math}
\begin{pmatrix}
1 & 3 \\
4 & 5
\end{pmatrix}
+
\begin{pmatrix}
A & g \\
e & \lambda
\end{pmatrix}
\end{math}
```

```
\mbox{$\mathfrak{R}$}_{1}=\mathbf{B}(\mathbb{C}^{2})\otimes I
```



Note this examples shows the normalization of font style, and insertion of a bounding box for the missing \otimes

```
\sin^2 \left(\pi+\theta\right)-\cos{\theta}+\tan{\lambda}=1
```



References

[1] S. MacLean and G. Labahn,, A Bayesian model for recognizing handwritten mathematical expressions, Pattern Recognition 48 (2015) 2433–2445.

[2] D. Zhelezniakov, V. Zaytsev and O. Radyvonenk, Acceleration of Online Recognition of 2D Sequences Using Deep Bidirectional LSTM and Dynamic Programming, in: International Work-Conference on Artificial Neural Networks, Springer, 2019, pp. 438–449.

[3] J. Zhang, J. Du and L. Dai, Track, Attend, and Parse (TAP): An End-to-End Framework for Online Handwritten Mathematical Expression Recognition, IEEE Transactions on Multimedia 21 (2018) 221–233.

[4] M. Mahdavi, R. Zanibbi, H. Mouchere, C. Viard-Gaudin and U. Garain, ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection, in: 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019, pp. 1533–1538.

[5] K. Davila, S. Ludi and R. Zanibbi, Using Off-Line Features and Synthetic Data for On-Line Handwritten Math Symbol Recognition, in: 2014 14th International Conference on Frontiers in Handwriting Recognition, IEEE, 2014, pp. 323–328.

[6] A. D. Le, B. Indurkhya and M. Nakagawa, Pattern Generation Strategies for Improving Recognition of Handwritten Mathematical Expressions, Pattern Recognition Letters 128 (2019) 255–262.

[7] V. T. M. Khuong, U. Q. Huy, N. Masaki and M. K. Phan, Generating Synthetic Handwritten Mathematical Expressions from a LaTeX Sequence or a MathML Script, in: 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019, pp. 922–927.

[8] R. Zanibbi and D. Blostein, Recognition and Retrieval of Mathematical Expressions, International Journal on Document Analysis and Recognition (IJDAR) 15 (2012) 331–357.

[9] R. Zanibbi, A. Aizawa, M. Kohlhase, I. Ounis, G. Topic and K. Davila, NTCIR-12 MathIR Task Overview, in: NTCIR, 2016.

[10] B. Chen, B. Zhu and M. Nakagawa, Training of an on-line handwritten Japanese character recognizer by artificial patterns, Pattern Recognition Letters 35 (2014) 178–185.

[11] C. Shorten and T. M. Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning, Journal of Big Data 6 (2019) 60.

[12] J.-W. Wu, F. Yin, Y.-M. Zhang, X.-Y. Zhang and C.-L. Liu, Image-to-Markup Generation via Paired Adversarial Learning, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2018, pp. 18–34.

[13] C. Wigington, S. Stewart, B. Davis, B. Barrett, B. Price and S. Cohen, Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network, in: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), volume 1, IEEE, 2017, pp. 639–645.

[14] Apostolos Syropoulos, Asana-Math, 2019. URL: https://ctan.org/tex-archive/fonts/Asana-Math/.

[15] Unicode Normalization Forms, 2021. URL: https://unicode.org/reports/tr15/.

[16] K. Davila, R. Zanibbi, A. Kane, and F. W. Tompa, Tangent-3 at the NTCIR-12 MathIR Task, in: NTCIR, 2016.

[17] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, in: Soviet Physics Doklady, volume 10, Soviet Union, 1966, pp. 707–710.

[18] B. R. Miller, A LaTeX to XML/HTML/MathML Converter (2020). URL: https://dlmf.nist.gov/LaTeXML/manual.pdf.

[19] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau,E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerk-wijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant,K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, Array programming with NumPy, Nature 585 (2020) 357–362.

[20] A. Graves, Generating Sequences with Recurrent Neural Networks, arXiv preprint arXiv:1308.0850 (2013).

[21] K. S. Tai, R. Socher and C. D. Manning, Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, arXiv preprint arXiv:1503.00075 (2015).

[22] E. Alonso, B. Moysset and R. Messina, Adversarial Generation of Handwritten Text Images Conditioned on Sequences, in: 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019, pp. 481–486.

[23] W.-D. Chang and J. Shin, A statistical handwriting model for style-preserving and variable character synthesis, International Journal on Document Analysis and Recognition (IJDAR) 15 (2012) 1–19.

[24] M. M. R. Alavi Milani, S. Hosseinpour and H. Pehlivan, Rule-Based Production of Mathematical Expressions, Mathematics 6 (2018) 254.