

Tools for hand-drawn corpus generation

S. MacLean

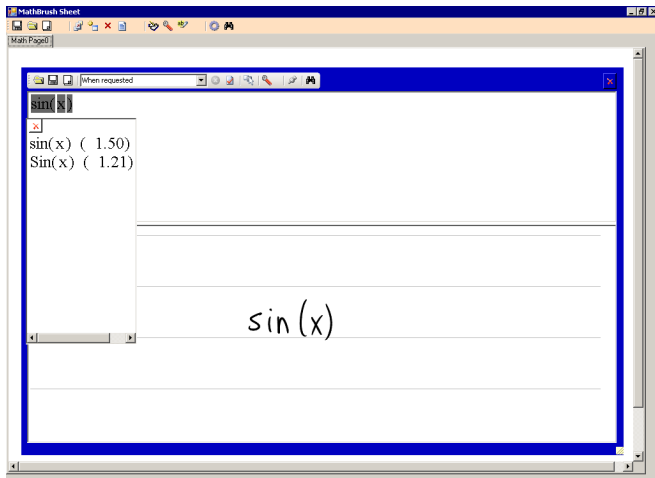
University of Waterloo

CICM 2009 @ Grand Bend

Outline

1. **Why** we created the corpus
 - ▶ MathBrush
2. **How** we created the corpus
 - ▶ Automatic template generation
 - ▶ Manual transcription
 - ▶ Automatic ground-truth labeling
3. **What** it can be used for

MathBrush



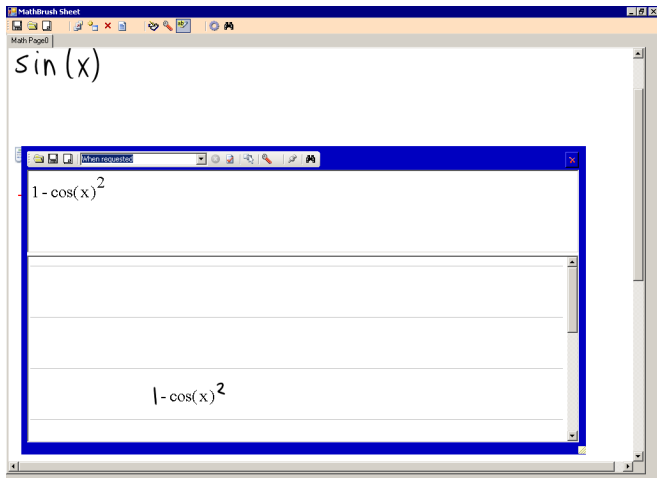
MathBrush

The screenshot shows a window titled "MathBrush Sheet" with a toolbar at the top. The main workspace contains two steps of a calculation:

[1] Ink Input
 $\sin(x)$

[2] Integrate [1] w.r.t. x
 $-\cos(x)$

MathBrush



Why a corpus?

1. Training

- ▶ Relational grammar approach for math recognition
- ▶ Need training data for relational classifiers

2. Testing

- ▶ No standard corpus for regression testing/recognizer comparison
- ▶ Important to identify weaknesses and evaluate performance

Creating the corpus

Three steps:

1. Generate a template expression
2. Transcribe the expression
3. Label the handwritten sample with ground-truth
 - ▶ symbol labels
 - ▶ symbol and subexpression relations

Template generation

Goal: randomly generate images containing template expressions

Advantages:

- ▶ infinite capacity
- ▶ unbiased sampling
- ▶ extensibility

Template generation

Process:

- ▶ model math using a context-free grammar
- ▶ generate a random derivation
- ▶ convert derivation $\rightarrow \text{\LaTeX} \rightarrow \text{image}$

Relational grammar

Similar to typical CFGs, but...

Productions of the form

$$A_0 \xRightarrow{r} A_1 A_2 \cdots A_k$$

where:

- ▶ A_0 is a nonterminal
- ▶ A_1, \dots, A_k are terminals or nonterminals
- ▶ r is a **relation** that must be satisfied by adjacent RHS tokens

Relational grammar

e.g.

$$[\text{ADD}] \overset{\rightarrow}{\Rightarrow} [\text{TERM}] + [\text{ADD}]$$
$$[\text{ADD}] \Rightarrow [\text{TERM}]$$
$$[\text{TERM}] \overset{\rightarrow}{\Rightarrow} [\text{VAR}] [\text{TERM}]$$
$$[\text{TERM}] \overset{\downarrow}{\Rightarrow} [\text{ADD}] \text{---} [\text{ADD}]$$
$$[\text{TERM}] \Rightarrow [\text{VAR}]$$
$$[\text{VAR}] \Rightarrow a|b|\dots$$

Random derivations

Naive **approach**:

1. pick a random production at current symbol
2. generate the RHS
3. recurse on nonterminal RHS tokens

Two **problems**:

1. Non-termination or very long expressions
2. Bias based on grammar structure

Controlling expression size

Problem: recursing on too many nonterminals

Solution: skip directly to terminals

- ▶ use parameter p_{inc}
- ▶ initialize value $p \leftarrow 0$
- ▶ instead of picking a random production:
 1. draw $x \in U[0, 1]$
 2. if $x < p$, derive a single terminal symbol if possible
 3. else, $p \leftarrow p + p_{inc}$ and pick a random production

Eliminating bias

Problem: randomly choosing productions gives non-uniform distribution over mathematics

Solution: randomly choose expression type instead of picking a random production:

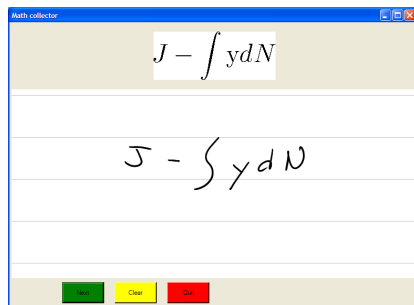
- ▶ if the current nonterminal can derive multiple expression types:
 1. choose one at random
 2. produce the appropriate derivation sequence
- ▶ else pick a random production

Generating templates

The whole process:

1. generate a parse tree as described
2. convert the parse tree to a \LaTeX string
3. convert \LaTeX to an image and display it for transcription

Collection software



- ▶ Template expression displayed as image
- ▶ Transcribed by study participant (x20)
- ▶ “Next” generates a new template

Automatic labeling

Doesn't this limit training capacity?

Not necessarily.

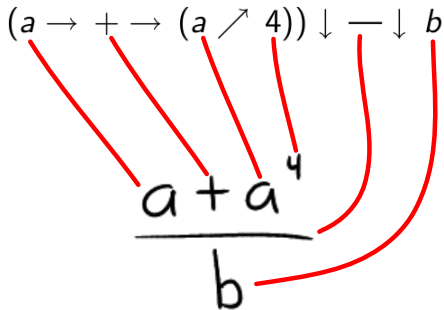
We have “extra” knowledge to ease labeling.

Use very simple labeling techniques, train sophisticated recognizers.

Automatic labeling

Match symbols and relations between a derivation and ink input

e.g.



Labeling technique

Idea: omit parentheses (subexpression groups) and match left-to-right

e.g.

$$a \rightarrow + \rightarrow a \nearrow 4 \downarrow - \downarrow b$$

- ▶ keep a running score measuring match goodness
- ▶ at each derivation symbol:
 1. measure relation membership to each unused recognition group
 2. order groups by recognizer and relation score
 3. recurse on each one until a total match is found

Experiments

Two scenarios:

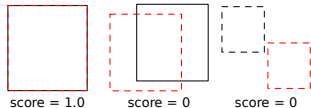
1. Run the algorithm on every sample.
2. Pre-train symbol recognizer randomly on 20% of each subject's samples, then run the algorithm on their samples.

Accuracy of labeling

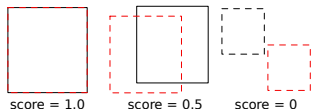
Agreement between auto-labeled symbol bounding boxes and manually-labeled bounding boxes.

3 measures of accuracy:

- ▶ Full expression: each expression assigned 0 or 1
 - ▶ assigned 1 iff every bounding box matches exactly
- ▶ Exact box: each box assigned 0 or 1
 - ▶ assigned 1 iff boxes match exactly



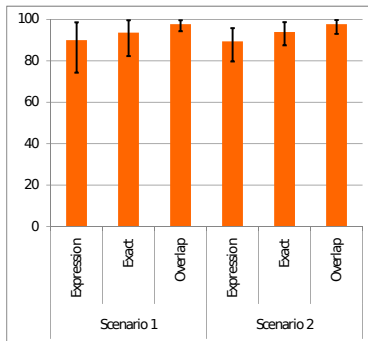
- ▶ Overlap: each box scores between 0 and 1
 - ▶ score is ratio of overlap area to larger box's area



Accuracy of labeling

Orange bars indicate average accuracy.

Error bars indicate range of accuracy over all 20 subjects



Average accuracy $\sim 90 - 98\%$

Reject rate $\sim 50\%$

Reject rate

Main problem is high reject rate.

- ▶ Scenario 1 only labeled $\sim 45\%$ of samples
- ▶ Pre-training only increased this to $\sim 55\%$

Largely due to simplicity of labeling technique

- ▶ but...must not sacrifice accuracy!

Future work

Recognition

- ▶ Training recognizers
- ▶ Evaluating recognizers
- ▶ Comparing recognizers

Ground truth

- ▶ Reduce reject rate
 - ▶ Subexpression context
 - ▶ Targetted pre-training

Corpus creation

- ▶ Expand corpus coverage
- ▶ Experiment with different semantic distributions
 - ▶ Using priors in math recognition

Conclusions

- ▶ Publicly-available corpus of math expressions
- ▶ Automatically-generated template expressions
 - ▶ Technique applicable to many domains
- ▶ Manually transcribed by hand
- ▶ Automatically labeled with ground truth
 - ▶ Highly-accurate approach
 - ▶ Simplicity leads to high reject rate

Thanks!

Corpus available at:

<http://www.cs.uwaterloo.ca/scg/mathbrush/mathdata/>