IUI Term Project: Equation 2 LaTeX

Osman Baskaya & Onur Kuru

May 23, 2014

Contents

1 Problem Definition

Writing an equation with a keyboard is hard and a boring task. When the equation is getting complicated, the author is more reluctant to write it. On the other hand, drawing an equation is much more natural and easier since we implicitly utilize two dimension rather than one we do in computer. Thus, converting a handwritten equation into the LaTeX form is important and it would be useful. Our project aims to recognize an equation and then convert it into its LaTeX equivalent form.

The roots of mathematical expression recognition problem go back to digit recognition problem. Digit recognition problem is an old problem and people came up with robust and accurate systems [Liu et al., 2003] with learning approaches as well as rule-based systems. [Bottou et al., 1994] analyzed the different classifiers on this problem.

Obtaining excellent performance in recognition of isolated digits and letters, people stared their eyes on the much harder problem: mathematical expression recognition. In order to recognize a mathematical expression, one should solve the digit and operator recognition problems. Research in mathematical expression recognition is also well-studied (see [Chan and Yeung, 2000] for details).

Recognition part can be categorized into two [Plamondon and Srihari, 2000]: (1) Online recognition uses an digital pen which collects pen-tip movements as well as the pen-up/pen-down switching. On the other hand, (2) offline recognition uses a scanned images of formulas. Offline recognition considered to be more difficult since it has less information such as hand action, stroke direction and so on.

In the next section, we will explain our system[1] which is based on offline recognition approach blended with machine learning and rule based decision mechanism.

2 System

2.1 System Overview

In this section we will give the parts of our system in detail. The system inputs an image file which contains an equation and then outputs the equation in LaTeX format. The main system can be seen as a pipeline composed of 4 parts. The inputed image goes through this pipeline and gets processed at each sub-system.

First part takes the image and does segmentation to locate the objects individually. Then these objects get fed into a classifier to recognize each as a digit (0-9) or an operator ($+, \times, \sum$.). When each object is classified accordingly, expression construction part takes the objects paired with their labels and outputs the equation as a string. The final part uses this expression string and transforms it to LaTeX.

[1]The code the replicate our findings can be found https://github.com/osmanbaskaya/equation2latex. It will be publicly available after presentation.

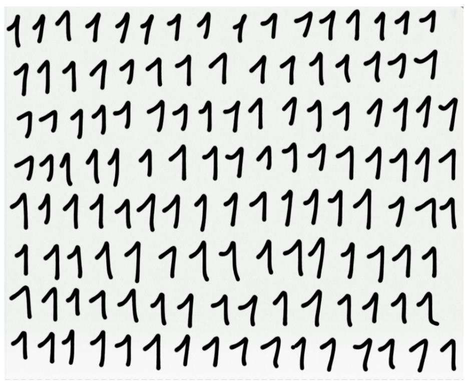


Figure 1: Input Example for Dataset Creation Tool

2.2 Datasets

MNIST dataset This dataset contains handwritten digit images, 60K examples for training and 10K examples for test. This images are not binary, range of the pixel values changes from 0 to 255.

Our dataset MNIST database[2] contains only digits, however, diversity of a mathematical expression is much richer. Therefore, we create our handwritten dataset. We follow the same procedure as in MNIST, except that our images are black and white, i.e., strokes are white (intensity equals to 255), background is black (intensity equals to 0). We used 3 annotators to draw digits and operators.

2.3 Dataset Creation Tool

In order to give an idea for generalization performance of a classifier, we drew some digits and created a small initial digit test set. When we trained our classifier using the training set of MNIST, we obtained low scores on our digit test set compared to MNIST test set, as seen in Table 1. Although we tried to follow the MNIST labelling procedure, we could not improve the performance on our prepared dataset. We gave considerable amount of time and tried to diagnose the performance difference between the MNIST test set and our initial digit test set. We examined the input of two dataset and made experiments with different parameters but, still, performance was significantly low. This is the motivation behind the creation of our own dataset not only for operators but also for digits.

To create a dataset quickly, we wrote a basic tool. It takes an image file which contains digits or operators, segments it and write each object into the given directory. Figure 1 shows an input (digit 1) for this toolbox.

2.4 Stroke Segmentation

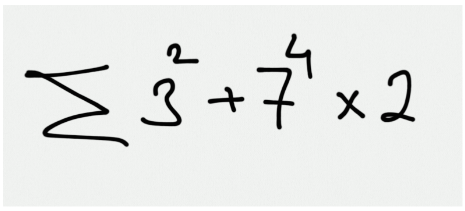
At the start of pipeline the input drawing (Figure 2a) goes through a segmentation where equation needs to be broken into pieces such as digits and operators. This part is summarized below:

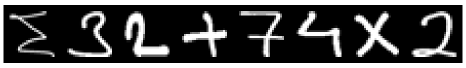
[2]http://yann.lecun.com/exdb/mnist/

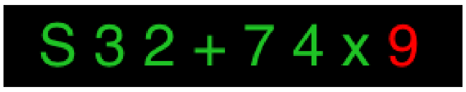
- Connectivity Analysis
- Scaling
- Padding

In order to segment strokes (i.e., digits and operators), we use connectivity analysis. We enquire 8-neighboring pixels since our black and white images are in 2-d. If white pixel has a white pixel in 8-neighboring distance, then this pixel is decided as connected with the first one.

Another problem needs to be solved is scaling. We follow the similar procedure as MNIST dataset. The size of each digit image is 28x28 in MNIST. However, digits and operators in mathematical expression can be any size. We scale and represent each digit and operator 21x21 matrix. After scaling, we use 3-pixel padding in order to center the object. Therefore, each digit or operator in mathematical expression is mapped into the same form as used in classifier training. An example segmentation is provided in Figure 2b.


(a) Drawing


(b) Segmented images



(c) Recognition

S3**2+7**4*9

(d) Expression

$\sum{3^{2}+{7^{4}}\times{9}}$

(e) Latex

Figure 2: Demonstration of the System Pipeline.

2.5 Digit and Operator Recognition

Softmax Regression The classification algorithm we used in this project is the softmax regression. Softmax regression can be seen as a logistic regression that is able to handle the cases where class labels can take more than two values. Our preliminary experiments (see Table 1) shows this classifier is powerful on digit recognition — accuracy of this classifier on MNIST is approximately 93%.

	M-train+M-test	M-train+Digits
Softmax	0.926	0.342
Softmax+Autoencoder	0.985	0.535

Table 1: Performance on different datasets

Softmax Regression with AutoEncoder Autoencoders are unsupervised learning algorithm that basically tries to find out dynamics of the input that is fed in. It may be thought of as a summarization function; it gets rid of all unnecessary part of the data (e.g. noise, redundant features) and puts forward the important patterns. It summarizes the input. Table 1 shows the vanilla softmax regression and softmax regression with autoencoder on various combinations of two datasets —MNIST and our prepared dataset. The softmax regression with autoencoder outperformed the vanilla version significantly on both MNIST and our prepared data. Therefore, we chose the softmax regression with autoencoder as our ultimate system and in the rest of the paper, all performance score are taken using this classifier algorithm.

At the recognition step, each segmented object gets classified with an integer label. For clarity in Figure 2c, we mapped the operator labels to characters where plus label was originally 11 but it is represented as +. We obtained %99 classification accuracy where we seperated our data as train and test set. The reported accuracy is the mean of 5 different runs.

2.6 Expression Construction

Following the image segmentation and recognition of each object individually, this subpart of the system utilizes the objects with their predicted labels to output a text expression (see Figure 2d). This part inputs the objects ordered by their positions on x-axis with their predicted labels. Labels represent digits or operators. Each object contains the information of their bounding box; upper (x,y) coordinates, width and height.

Our algorithm makes a pass on the ordered objects one by one and maps integer labels to a character. For instance $\sum$, $+$, $\times$ are mapped to S,+,* and digits are mapped to ASCII characters. These characters are further used in composition of the whole expression. The power operation is detected at this part, outputted as **. The detection of power operation is determined according to the relative positions of consecutive digits.

2.7 Expression to Latex Transformation

Final part of the system translates the expression string to LaTeX equivalent (see Figure 2e). This transformer expects a legitimate mathematical expression. To realize the conversion, we used the compiler[3] module of python to parse the text expression. This module has a parse method that takes the string equation and returns a parse tree. Each node of the parse tree represents some part of the equation. Leaf nodes represent constants that are used in the equation. This parse tree is constructed according to the operator precedence.

Our algorithm locates the position of $\sum$ and divides the equation to halves. Then for each sub-equation, a parse tree is constructed by using the compiler module. Each parse tree is processed recursively and the code maps the operators to equivalent LaTeX formats. Walking the parse tree recursively makes the code obey the operator precedence.

2.8 Limitations

Although we described our system in detail, we glossed over some of the limitations associated with our system. To make the dataset creation less labor-intensive and the overall problem more managable, we decided to restrict the input to all the digits (0-9) and to only three operators which are sigma, plus and times. The user can also take the power of any number. Numbers in the equation are not limited to single digits, however if the number is used in the power part, it has to be a single digit. Unfortunately, the sigma cannot be written with lower and upper bounds and more than one sigma cannot be used in the same equation.

3 System Evaluation

In this section, we explain our evaluation methodology and discuss the system performance on equation and object recognition levels. Performance of the system strictly depends on the recognition part, because the other parts of the system is built as rule-based. Therefore we followed two evaluation strategies to measure the performance of our system.

Equation Recognition Evaluation We reported the accuracy of equation recognition whether the system could recognize the equation completely. If the system could completely translate the equation in the drawing to LaTeX, we accepted that the system classified that image correctly. Therefore the equation accuracy reports the percentage of images correctly transformed.

	Equation Accuracy	Object Accuracy
Ozan	0.00	0.50
Volkan	0.30	0.83
Engin	0.20	0.60
Guray	0.20	0.73

Table 2: Evaluation

[3]https://docs.python.org/2/library/compiler.html

Object Recognition Evaluation To be able to translate the equation in the image to LaTeX form, the image goes through the pipeline where each object in the image is classified individually. If one of the objects gets misclassified then the whole equation becomes invalid. Even more, the equation may not be legitimate mathematical expression where the system cannot proceed to translate it to LaTeX. Since the equation recognition evaluation is a bit harsh, we wanted to report what percentage of the objects classified correctly.

For measuring the performance of our system, we asked four different people to draw 10 equations each. This yielded 40 equations containing 230 objects overall. Performance results are presented in Table 2. We argued that the equation accuracy was a harsh metric and from the results we can read the same. Our system performed very well on our object data when tested with different folds, however, it could not achieve the same performance when the objects were taken from equation of other people. Overall, it achieved 0.67 object accuracy and 0.18 equation accuracy. There is a significant drop of performance.

When we inspected the equations, we could see that our data was not enough for generalizing to the inputs from random people because everybody has different style of writing digits and symbols. The major difference between drawing objects was on ones and sevens. This clearly shows that one should get drawings from many different people to construct a dataset for generalizing well. On the other hand, the performance on our dataset shows that, with enough drawing data, we could achieve promising results.

4 Conclusion

In this document, we proposed a system that converts a handwritten mathematical expression to equivalent equation form in LaTeX. We explained our pipeline in detail and the limitation of the system. Although our proposed system manages only small set of operators, it is not difficult to add other operators with similar functionalities (such as $\prod$ and $\int$). The overall object recognition performance of our system is 0.67 which is promising whereas the equation recognition performance is low. As we mentioned, our equation recognition performance evaluation might be harsh since any misclassified object (digit or operator) makes equation recognition incorrect. One simple improvement would be to expand the dataset using more annotators and different sized object images. We used similar sized objects on training data (see Figure 1) which, in retrospect, was wrong decision. For future work, we will feed our classifier with various sized digits and operators. Another improvement one can make is to increase robustness of the rule-based part of the system. Currently, the expression construction part utilizes only the bounding-box coordinates of consecutive objects. It will be more robust to make this part not only using the coordinates of two consecutive objects but also taking into account radial angle between objects.

References

[Bottou et al., 1994] Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Jackel, L. D., LeCun, Y., Muller, U. A., Sackinger, E., Simard, P., et al.

(1994). Comparison of classifier methods: a case study in handwritten digit recognition. In *International Conference on Pattern Recognition*, pages 77–77. IEEE Computer Society Press.

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

[Liu et al., 2003] Liu, C.-L., Nakashima, K., Sako, H., and Fujisawa, H. (2003). Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern Recognition*, 36(10):2271–2285.

[Plamondon and Srihari, 2000] Plamondon, R. and Srihari, S. N. (2000). On-line and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):63–84.